



## **ElectricInsight 4.0 User's Guide**

**Electric Cloud, Inc.**  
35 South Market Street, Suite 100  
San Jose, CA 95113  
[www.electric-cloud.com](http://www.electric-cloud.com)

## **ElectricInsight 4.0**

Copyright © 2002–2014 Electric Cloud, Inc. All rights reserved.

Published 9/22/2014

Electric Cloud® believes the information in this publication is accurate as of its publication date. The information is subject to change without notice and does not represent a commitment from the vendor.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” ELECTRIC CLOUD, INCORPORATED MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any ELECTRIC CLOUD software described in this publication requires an applicable software license.

Copyright protection includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including without limitation, material generated from software programs displayed on the screen such as icons, screen display appearance, and so on.

The software and/or databases described in this document are furnished under a license agreement or nondisclosure agreement. The software and/or databases may be used or copied only in accordance with terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement.

### **Trademarks**

Electric Cloud, ElectricAccelerator, ElectricCommander, ElectricInsight, and Electric Make are registered trademarks or trademarks of Electric Cloud, Incorporated.

Electric Cloud products—ElectricAccelerator, ElectricCommander, ElectricInsight, and Electric Make—are commonly referred to by their “short names”—Accelerator, Commander, Insight, and eMake—throughout various types of Electric Cloud product-specific documentation.

Other product names mentioned in this guide may be trademarks or registered trademarks of their respective owners and are hereby acknowledged.

# Contents

<b>Chapter 1: ElectricInsight Introduction</b>	<b>1-1</b>
About ElectricInsight	1-1
<b>Chapter 2: Installing ElectricInsight</b>	<b>2-1</b>
Install Using the GUI	2-1
Install Using the Console	2-2
Installation Options	2-2
<b>Chapter 3: Getting Started</b>	<b>3-1</b>
Annotation	3-2
Starting ElectricInsight	3-2
Navigating the Interface	3-3
Agents & Jobs	3-3
Navigation	3-5
Overview	3-5
Legend	3-6
Job Summary	3-6
Zoom	3-7
Replay	3-7
Monitor Live Build	3-8
<b>Chapter 4: Build, Job, and Make Information</b>	<b>4-1</b>
Build Properties	4-2
Job Details Dialog	4-2
Job Details	4-3
Job Path	4-3
Annotation	4-4
Environment	4-5
Submakes	4-5
Make Details Dialog	4-6
Make Details	4-6
Make Jobs	4-7
Make Path	4-8
Searching for Jobs	4-8
Search Examples	4-9
Supported Search Fields	4-10
Viewing All Jobs	4-11
Using the Hyperlog	4-12
<b>Chapter 5: Reports</b>	<b>5-1</b>
Build Manifest	5-2

Build Metrics .....	5-2
Build Summary .....	5-3
Cluster Sharing .....	5-3
Derived Files Analysis .....	5-4
ElectricSimulator .....	5-5
Export Timeline .....	5-7
Files Modified Multiple Times .....	5-7
Job Stats .....	5-8
Job Time by Type .....	5-9
Jobs by Agent .....	5-10
Jobs by File .....	5-11
Longest Jobs .....	5-11
Longest Serial Chain .....	5-12
Makefile Manifest .....	5-13
Most Read Files .....	5-14
Root Conflicts .....	5-15
Serialization Analysis .....	5-15
Command-Line Enabled Reports .....	5-16
Creating a Custom Report .....	5-18
<b>Chapter 6: Understanding Build Performance .....</b>	<b>6-1</b>
<b>Appendix A: Annolib Programmer's Reference .....</b>	<b>A-1</b>

# Chapter 1: ElectricInsight Introduction

Until now, there has been little visibility into builds to “see” why a build was slow, why a build broke, or which dependencies were involved. ElectricInsight® (Insight) removes the “black box” around software product builds and provides easy-to-understand performance data.

## About ElectricInsight

Insight depicts how a build is structured and run, empowering build managers to pinpoint performance problems or conflicts in a parallel build. Developed to work with ElectricAccelerator®, Insight mines information produced by Electric Make® to provide an easy-to-understand, graphical representation of the build structure for performance analysis.

Insight provides detailed information and reports for at-a-glance diagnostics for each job on each host in the build cluster. Users can identify which jobs are performing, when, and with which files. Instead of manually reading through tens of thousands of lines of information in log files, error detection and performance tuning that used to consume hours or days can now occur in a few minutes or seconds.

By observing and tracing serialization sources or slowdowns, you can fine tune your build for maximum speed. Insight gives you the ability to pinpoint areas to improve in your build process—and you will have answers for these questions:

- Are any unnecessary serializations occurring?
- Is there a gap where one or more agents is not busy?
- Which job chains are the longest and can they be shortened?
- Which files are being modified (created, updated, or deleted) multiple times?

## Chapter 2: Installing ElectricInsight

Because Insight analyzes build annotation files, it is simplest to install it on the machine where you run Electric Make, or on a Cluster Manager host where you can upload build annotation files.

The Insight distribution consists of the following files (Windows/Linux):

- `einsight.exe/einsight` - the ElectricInsight program
- `annolib.dll/annolib.so` - the annolib library

### Install Using the GUI

To install Insight on Windows or Linux using the GUI, follow this procedure:

1. Run the appropriate install file to begin installation:

For Windows: `ElectricInsight-<version>-Windows-Install.exe`

You *must* run the installer as administrator. For example, on Windows systems running Windows 2008 or later, right-click the installer and select Run as administrator.

For Linux: `ElectricInsight-<version>-Linux-x86-Install`

2. Click **Next** on the Welcome screen.

3. Accept the default installation location:

- Windows - `C:\ECloud\i686_win32\bin`

- Linux - `/opt/eccloud/i686_Linux/bin`

Or click **Browse** to select a different location, and then click **Next**.

4. Click **Browse** to locate your license file, and click **Next** to continue.

If you choose to provide the license file later, ensure you name it `license.xml` and copy it to the appropriate location in the Insight installation directory:

- For Windows: `C:\ECloud\i686_win32\bin`

- For Linux: `/opt/eccloud/i686_Linux/bin`

5. If the installer does not detect a PDF viewer, you are reminded to obtain one. Click **Next**.
6. Review your settings before continuing the installation. Click **Next** to continue or **Back** to make changes.

Your installation is complete.

7. Click **Finish** to close the installer.

**Note:** The Windows installation log is located in the following hidden directory: C:\Program Files\InstallJammer Registry\

## Install Using the Console

To install Insight on Linux using the console, follow this procedure (assumes you do not use any installation options):

1. Run the install file: `ElectricInsight-<version>-Linux-x86-Install`
2. Accept the default install location, `/opt/ecloud/`, or designate a different location.
3. Type the location of your license file.
4. If the installer does not detect a PDF viewer, you are reminded to obtain one.
5. Press **Enter** to continue the installation.

Your installation is complete.

## Installation Options

If you wish to launch the installer with options, use this format:

```
<installer filename> [options]
```

You can use multiple options to run an installation with customized settings and minimal interaction. The following installation example runs in silent mode, designates the license file location, and sets a different directory prefix:

```
ElectricInsight-4.0.0.0-Windows-Install.exe /mode silent /licensefile C:\mylicenses  
/prefix C:\ecloud2
```

The following options are available to customize your installation:

Option	Description
console	Run the installation in the console (not the GUI).
help	Display help information.
licensefile	Set the license file that the installer imports. Type the full path to the license file. Example: ElectricInsight-4.0.0.0-Windows-Install.exe /licensefile C:\mylicenses
mode	Set the installation mode. The following modes are available: <ul style="list-style-type: none"><li>• standard - the standard installation mode, which includes all user prompts</li><li>• silent - runs the installation without user prompts</li><li>• default - runs the installation while accepting all installation defaults, resulting in fewer user prompts</li></ul>
prefix	Set the installation directory. Type the full path. Example: ElectricInsight-4.0.0.0-Windows-Install.exe /prefix C:\ecloud2
temp	Set the temporary directory used by the program.

Option	Description
<code>version</code>	Display installer version information.



## Chapter 3: Getting Started

The following sections discuss what you need to know to start using Insight.

**Topics:**

- [Annotation](#)
- [Starting ElectricInsight](#)
- [Navigating the Interface](#)

## Annotation

ElectricInsight gathers information from build annotation files to create a picture of your build. To use Insight, you must first run your builds with the `--emake-annodetail` parameter.

For example, on the eMake command line, run:

```
% emake ... --emake-annodetail=file,waiting,history ...
```

**Note:** Collecting annotation information may cause your build to run approximately 5% slower.

### Supported Annotation Detail Levels

basic	Detailed information about each “job” in the build, including command arguments, output, exit code, timing, and source location
env	Environment variable modification information
file	Filesystem operations information, excluding lookups
history	Information about implicit dependencies discovered by Electric Make
lookup	Filesystem operations information, including lookups
md5	MD5 checksums for files read and written by the build
registry	Registry operations information
waiting	Information required to reconstruct the complete dependency graph

**Note:** The “Annotation” chapter in the Electric Cloud Electric Make Users Guide contains additional information about annotation.

### Annotation File Splitting

Because of limitations in ElectricInsight (a 32-bit application), eMake automatically partitions annotation files into 1.6 GB “chunks.” The first chunk is named using the file name that you specify with the `--emake-annofile` option or with “emake.xml,” if `--emake-annofile` is not specified. The second chunk uses that name as the base but adds the suffix `_1`, the third chunk adds the suffix `_2`, and so on. For example, a four-part annotation file might consist of files named `emake.xml`, `emake.xml_1`, `emake.xml_2`, and `emake.xml_3`.

No special action is required to load a multipart annotation file into ElectricInsight. If all parts are present in the same directory, ElectricInsight automatically finds and loads the content of each file—simply specify the name of the first chunk when opening the file in ElectricInsight.

## Starting ElectricInsight

To start ElectricInsight:

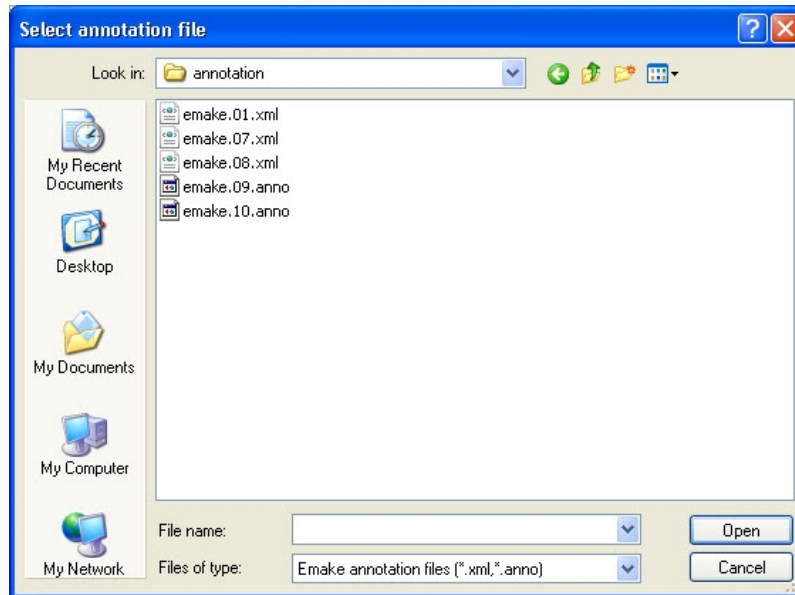
1. Go to Start > All Programs > Electric Cloud > ElectricInsight

or

Type `einsight` at the command prompt. Optionally, you can specify the target annotation file.

```
% insight [build-annotation-file]
```

- If you did not specify a build annotation file on the command line, click the folder icon on the toolbar or select File > Load annotation.



- Navigate to the annotation file of a completed build that you want to analyze and click **Open**.

Your build is now displayed.

## Navigating the Interface

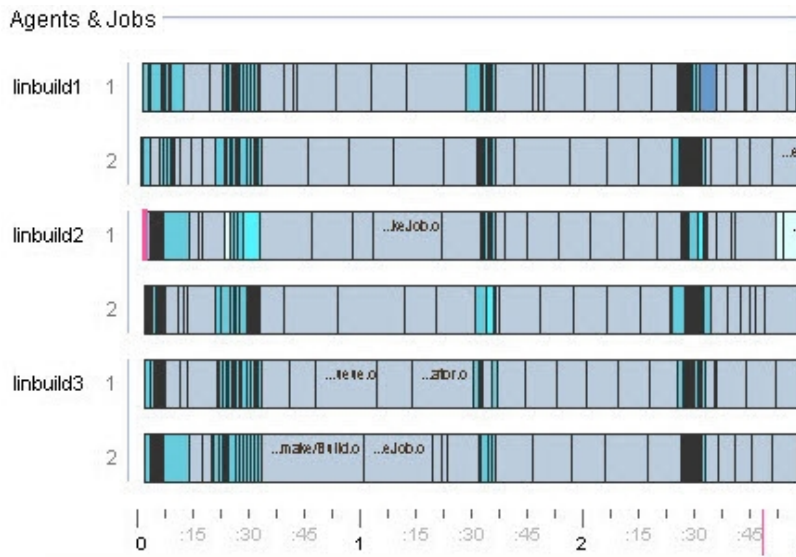
This section will familiarize you with the various portions of Insight's navigation.

### Topics:

- [Agents & Jobs](#)
- [Navigation](#)
- [Overview](#)
- [Legend](#)
- [Job Summary](#)
- [Zoom](#)
- [Replay](#)
- [Monitor Live Build](#)

## Agents & Jobs

In the Agents & Jobs section (outlined in the following screenshot), each agent used in the build is represented by one row. Host names and agent number designations appear to the left of the agent job bars. Use the view frame and arrows in the [Navigation](#) section to view agents that are not currently visible.



The section's x-axis is time, with the build starting at the left. The first job completed by each agent is at the extreme left of an agent's job bar. As you scroll to the right on an agent's job bar, you see the progress of jobs assigned to the agent.

The time grid helps you determine how long jobs take. The grid has major and minor grid lines. By default, major grid lines are 60 seconds (pixels) apart and minor grid lines are 15 seconds (pixels) apart. Build magnification is measured in pixels per second (pps). As you zoom in or out, this ratio increases or decreases. Magnification is displayed in the lower right corner.

To see how long some jobs took to complete, click and drag from the left edge of the first job to the end of the last job of interest. The job ruler appears and displays a time measurement. You can also use the ruler by dragging from right to left.

### ***Dependencies and Waiting Jobs Popup***

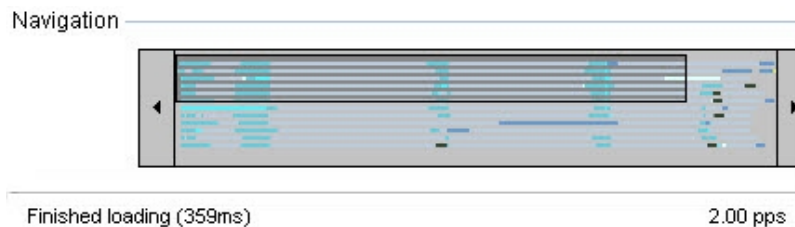
You can right-click a job and select Show dependencies or Show waiting jobs. Dependencies are outlined in blue. Waiting jobs (not present in the screenshot) are outlined in red.



For dependency details between a pair of jobs, load an annotation file that contains waiting and history level annotation and run the [Serialization Analysis](#) report.

## Navigation

In the Navigation section, the view frame indicates which part of the build the [Agents & Jobs](#) section currently displays. If the entire build is displayed, the view frame is the same size as the scroll bar. Drag the frame to view different build stages. You can also use the arrows on either side of the navigation bar to move the view frame.



Below the Navigation section is a field showing how long it took Insight to load build annotation information.

## Overview

The Overview section displays the following build information:

- Build ID and whether it was successful
- Build duration
- Number of makes in the build
- Number of agents and hosts used by the build
- Number of jobs in the build
- Annotation details collected in the build



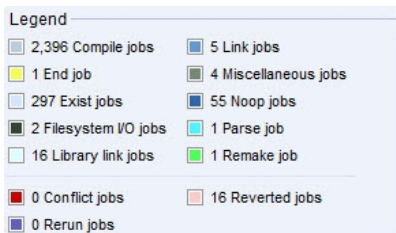
Keep in mind the following terminology:

- Each makefile has one or more *rules* (lines of text).
- A *target* is the *rule* output.
- A *command* is a single shell invocation in a *rule*.
- A *job* corresponds to a *rule* scheduled as part of a build—in most cases.
- There are jobs that parse a makefile. Jobs also have status—possible status values include *normal*, *conflict*, *rerun*, *reverted*, and *skipped*. For more details about job status, see [Supported Search Fields](#).

**Note:** You can hide the Overview, Legend, and Job Summary sections, by clicking the gray bar between those sections and the Agents & Jobs section. Click it again to show the hidden sections.

## Legend

The Legend section displays the total number of jobs for each type and status within the build. Click a job type or status to list all of its instances within the build. Legend colors correspond to the job type and status for individual jobs within agent job bars.



The [Agents & Jobs](#) section displays two dimensions of job data—job status and job type. If the job status is not unusual, then only the job type (such as parse or rule) is displayed. If a job has an unusual status (such as conflict or reverted), that color is used for the lower portion of the job and the upper portion retains the color of the job type.

## Job Summary

To display a job's summary information, mouse over any job in an agent job bar. The selected job is outlined in pink in the agent job bar.

The Job Summary section displays the following information about the selected job:

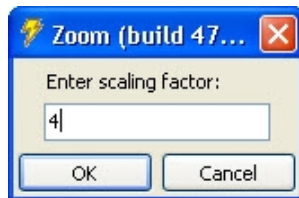
- Job ID and job type or status (for additional information about job types and status, see [Supported Search Fields](#))
- Start time (as an offset from the build start time)
- End time
- Job length in hours/minutes/seconds
- Output target name, if applicable

Job Summary	
Job:	<b>J082de170</b> (parse)
Start:	2.35s
End:	2.85s
Length:	0.49s
Target:	
	<no name>

## Zoom

The Tools menu contains the following magnification options:

- Zoom in and Zoom out - You can also zoom in or out using the magnifying glass icons in the toolbar or by pressing CTRL-[equal sign] or CTRL-[minus sign].
- Zoom to... - This option allows you to specify a magnification factor for the display. To return to the initial, unzoomed view, select Tools > Zoom to... and type a zoom factor of 1.0.



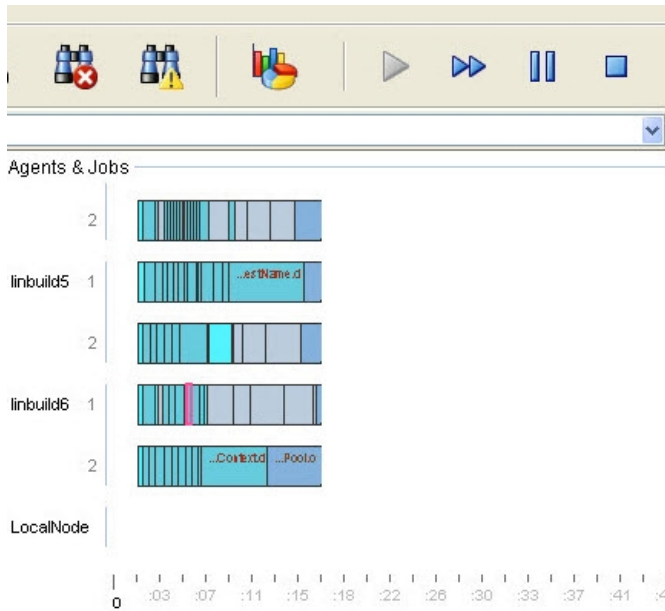
- Zoom to fit - This option sizes the display to fit the available space. Doing this may leave empty space to the left of all agent job bars, which depicts the time when eMake was parsing the makefile, before agents were assigned jobs. It is also possible that agents you are not currently viewing were working while agents you are viewing were not assigned to this build.

## Replay

Insight enables you to replay your build in the [Agents & Jobs](#) section so you can observe its progress.

You can display the replay toolbar by clicking Replay > Show.

The following screenshot shows the replay toolbar and a build replay in progress.



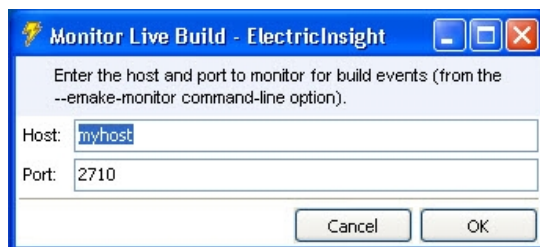
Replay controls:

- Play - starts the replay.
- Fast forward - increases replay speed. You can click fast forward up to three times to speed up the replay. Clicking a fourth time returns the speed to normal.
- Pause - pauses the replay. Click play to start again.
- Stop - stops the replay.

## Monitor Live Build

Insight enables you to monitor a live build in the [Agents & Jobs](#) section. Follow these steps:

1. Select File > Monitor live build to display the following dialog.



2. Provide the host and port information from the `--emake-monitor` command-line option:
  - **Host** is the hostname/IP of the Insight machine where the build will be monitored.
  - **Port** is the port number of the Insight machine that listens for the data.
3. Click **OK**.
4. Make sure `--emake-monitor=<hostname/IP>:<port>` is added to the command line.
5. Start the build.



**Note:** To monitor a live build, you must launch the Insight live build monitor **before** you start the build.

## Chapter 4: Build, Job, and Make Information

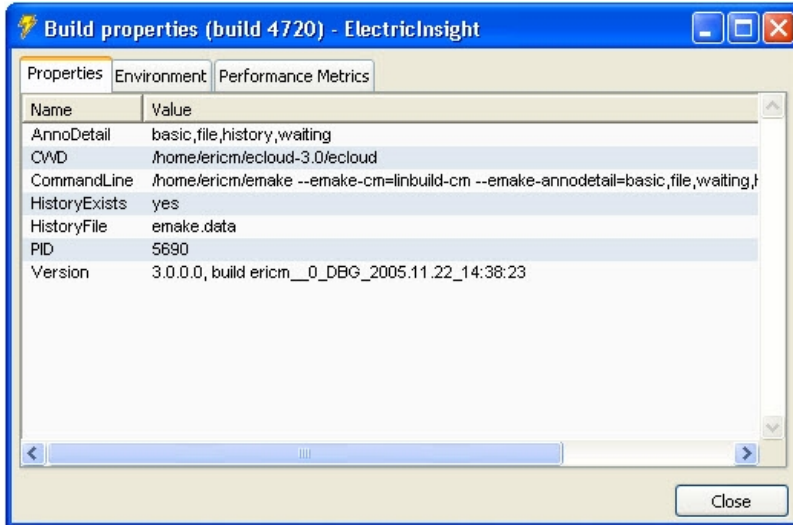
This section discusses how to view detailed build, job, and make information.

### Topics:

- [Build Properties](#)
- [Job Details Dialog](#)
- [Make Details Dialog](#)
- [Searching for Jobs](#)
- [Viewing All Jobs](#)
- [Using the Hyperlog](#)

## Build Properties

To view overall build properties, click File > Build Properties. Three tabs are available: Properties, Environment, and Performance Metrics.



The Environment tab displays build environment information, which you can copy to the clipboard:

1. Right-click anywhere in the window.
2. Select **Copy to clipboard**.
3. Select **...for bash/ksh** or **...for cmd**.



## Job Details Dialog

To open the Job Details dialog, double-click a job in the Agents & Jobs section. Three tabs are always available: Job Details, Job Path, and Annotation. The Submakes tab is available if the job includes submakes. The Environment tab is available if you specified the `env` annotation level.

### Topics:

- [Job Details](#)
- [Job Path](#)
- [Annotation](#)
- [Environment](#)
- [Submakes](#)

## Job Details

This tab displays summary information about the job. The following fields contain the same information as they do on the main screen: ID, Type, Start, End, and Length.

Job Details: jb36c7c1c .../emake (build 4720) - ElectricInsight

Job Details | Job Path | Annotation

ID: jb36c7c1c Type: rule Serial order: 796

Start: 114.17369s End: 156.010421s Length: 41.836731s Agent: linbuild5-1

Target name: ../i686\_Linux/ecldoud/emake/emake

Working dir: /home/ericm/ecldoud-3.0/ecldoud/emake

Makefile: ../config/rules.mak:187-190

Output: ☐ Show commands

```
g++ -pthread -g -Wall -Wno-unknown-pragmas -DUSE_PROFILING -DUSE_LOGGING -DHAVE_UNIX -DHAVE_LINUX -I../usr/local/tools/common/include -I../usr/local/tools/common/include/boost-1_32 -I../usr/local/tools/i686_Linux/curl-7.14.0/include -DEMULATE_GNU_MAKE -Wl,-rpath=/opt/ecldoud/i686_Linux/lib ../i686_Linux/ecldoud/emake/AgentManager.o ../i686_Linux/ecldoud/emake/Annotate.o ../i686_Linux/ecldoud/emake/AnnotationUploader.o ../i686_Linux/ecldoud/emake/ArchiveTarget.o ../i686_Linux/ecldoud/emake/BackingStore.o ../i686_Linux/ecldoud/emake/Build.o ../i686_Linux/ecldoud/emake/CacheInfo.o ../i686_Linux/ecldoud/emake/Chain.o ../i686_Linux/ecldoud/emake/ClusterManager.o ../i686_Linux/ecldoud/emake/CommandJob.o ../i686_Linux/ecldoud/emake/Content.o ../i686_Linux/ecldoud/emake/ContinuationJob.o ../i686_Linux/ecldoud/emake/DirCache.o ../i686_Linux/ecldoud/emake/Directory.o ../i686_Linux/ecldoud/emake/EmakePipelineStages.o ../i686_Linux/ecldoud/emake/EndJob.o ../i686_Linux/ecldoud/emake/Event.o ../i686_Linux/ecldoud/emake/Exceptions.o ../i686_Linux/ecldoud/emake/ExistenceJob.o ../i686_Linux/ecldoud/emake/FileData.o ../i686_Linux/ecldoud/emake/FileSystem.o ../i686_Linux/ecldoud/emake/FollowJob.o ../i686_Linux/ecldoud/emake/History.o ../i686_Linux/ecldoud/emake/Job.o ../i686_Linux/ecldoud/emake/JobQueue.o ../i686_Linux/ecldoud/emake/KSCommands.o ../i686_Linux/ecldoud/emake/LocalClusterManager.o ../i686_Linux/ecldoud/emake/LocalNode.o ../i686_Linux/ecldoud/emake/Logger.o ../i686_Linux/ecldoud/emake/Make.o ../i686_Linux/ecldoud/emake/MakePlatform.o ../i686_Linux/ecldoud/emake/Monitor.o ../i686_Linux/ecldoud/emake/Name.o ../i686_Linux/ecldoud/emake/Node.o ../i686_Linux/ecldoud/emake/Operation.o ../i686_Linux/ecldoud/emake/ParseJob.o ../i686_Linux/ecldoud/emake/Pattern.o ../i686_Linux/ecldoud/emake/PatternTable.o ../i686_Linux/ecldoud/emake/Preprocessor.o ../i686_Linux/ecldoud/emake/RelocTable.o ../i686_Linux/ecldoud/emake/RemakeJob.o ../i686_Linux/ecldoud/emake/RemoteClusterManager.o ../i686_Linux/ecldoud/emake/RemoteNode.o ../i686_Linux/ecldoud/emake/Result.o ../i686_Linux/ecldoud/emake/Rule.o ../i686_Linux/ecldoud/emake/Target.o ../i686_Linux/ecldoud/emake/TargetChain.o ../i686_Linux/ecldoud/emake/Terminator.o ../i686_Linux/ecldoud/emake/Utilities.o ../i686_Linux/ecldoud/emake/Variable.o ../i686_Linux/ecldoud/emake/VariableContext.o ../i686_Linux/ecldoud/emake/VariableTable.o ../i686_Linux/ecldoud/emake/VPathTable.o ../i686_Linux/ecldoud/emake/Version.o ../i686_Linux/ecldoud/emake/Worker.o ../i686_Linux/ecldoud/emake/GMake.o ../i686_Linux/ecldoud/emake/GMakePreprocessor.o ../i686_Linux/ecldoud/emake/main.o ../i686_Linux/ecldoud/compressor/compressor.a ../i686_Linux/ecldoud/util/util.a ../i686_Linux/ecldoud/cmconnect/cmconnect.a ../i686_Linux/ecldoud/xmlparser/xmlparser.a ../i686_Linux/ecldoud/tinyhttp/tinyhttp.a \
-o ../i686_Linux/ecldoud/emake/emake \
../i686_Linux/ecldoud/compressor/compressor.a ../i686_Linux/ecldoud/util/util.a ../i686_Linux/ecldoud/cmconnect/cmconnect.a ../i686_Linux/ecldoud/xmlparser/xmlparser.a ../i686_Linux/ecldoud/tinyhttp/tinyhttp.a
```

Go to job Close

For some job types, the **Target name**, **Makefile**, and **Output** fields are populated.

If the **Output** field is populated, you can enable **Show commands** to see the makefile line(s) that correspond to the job.

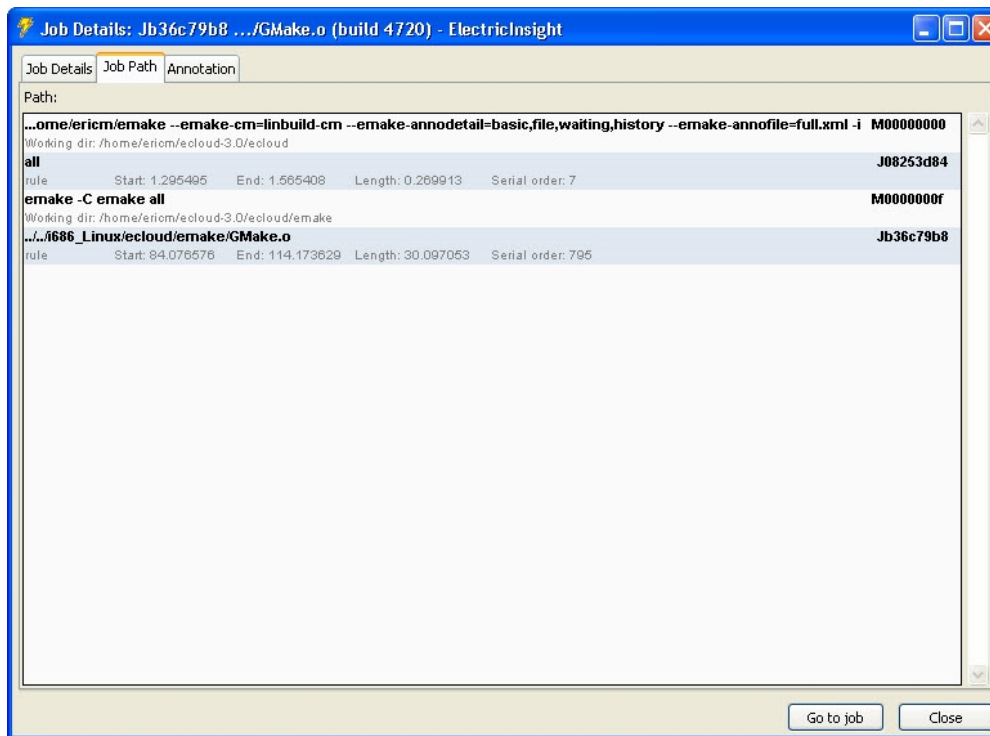
The **Makefile** field identifies the makefile and the relevant line in that makefile that created the job. You can click the makefile to open it in an editor. The **Makefile** field is empty for parse-type jobs because parse jobs are not created from a line in a makefile. For example, there is no rule in the makefile that instructs eMake to parse the makefile. To find which makefile a job parses, refer to the Directory and Name columns in the [Job Path](#) tab.

You can search the Output field by pressing CTRL-F. Pressing CTRL-G finds the next occurrence.

## Job Path

This tab displays the path to the makefile line(s) that created the job.

The first line contains the command that invokes the top level makefile. Typically, this line invokes another makefile. If it does, the second line displays the job ID. If it does not invoke another makefile, the second line displays the name of the target the job executes.

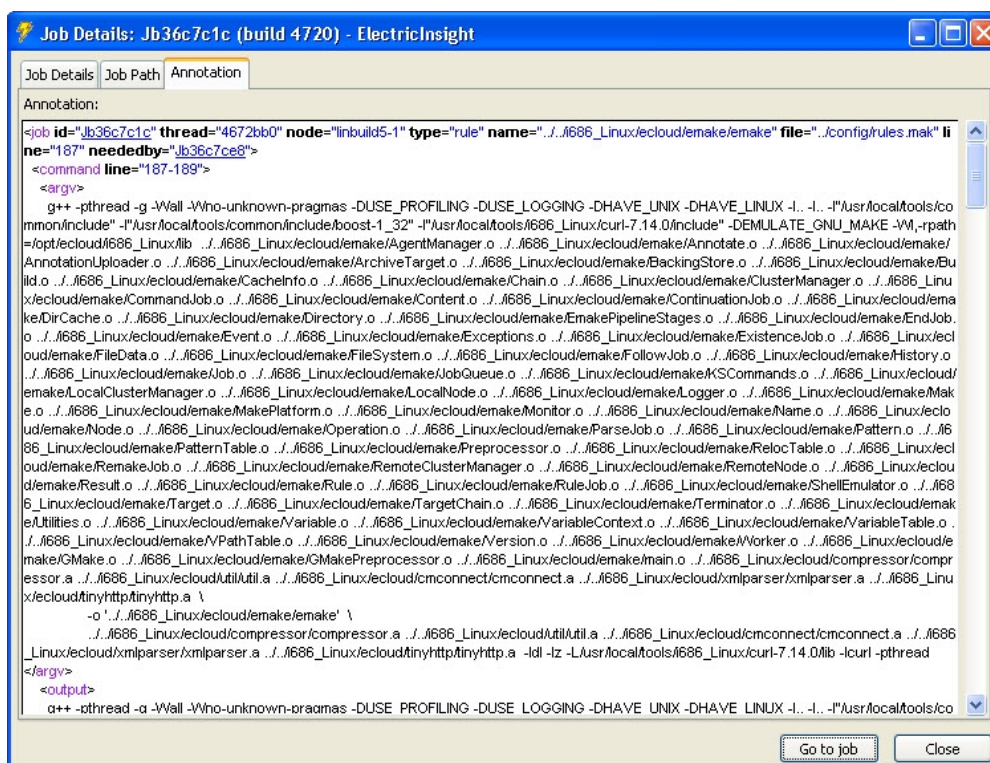


If the makefile line that created the job invoked a second makefile, this pattern of makefile lines and job IDs is repeated, chaining through makefiles until the target that created the job is reached.

## Annotation

This tab displays the portion of the annotation file pertaining to the job.

Pressing CTRL-F searches for and highlight text; CTRL-G finds the next occurrence.



## Environment

This tab displays environment information that was used to run the job. The Environment tab is available only if your annotation file includes `env` level annotation.

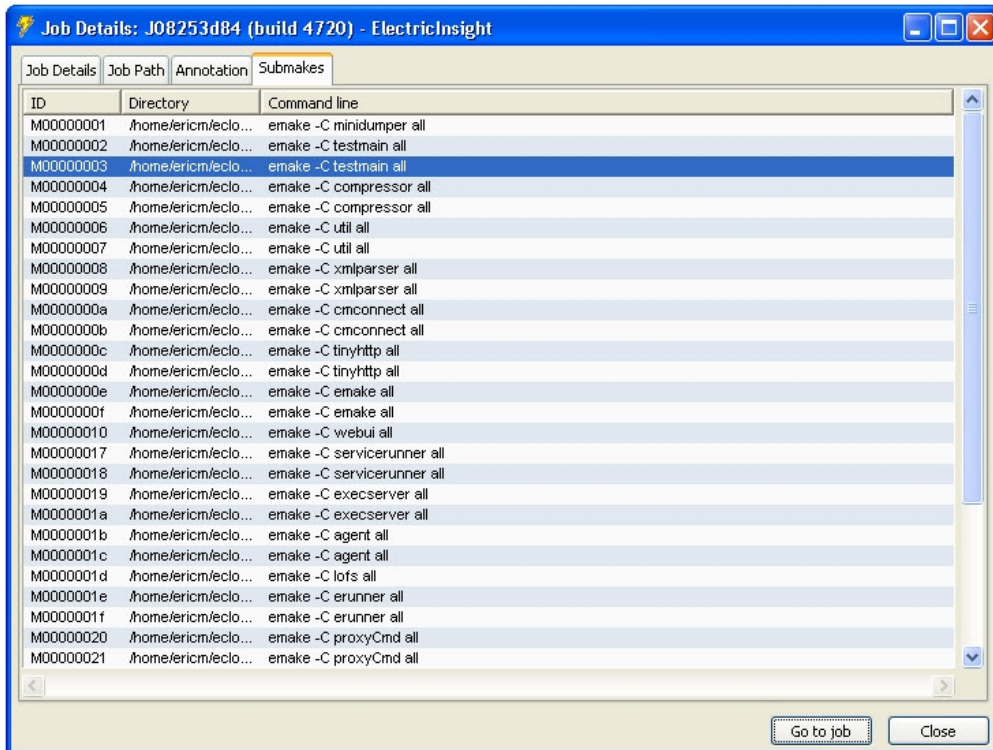
You can copy environment information to the clipboard:

1. Right-click anywhere in the window.
2. Select **Copy to clipboard**.
3. Select **...for bash/ksh** or **...for cmd**.

## Submakes

This tab lists all submakes that the job runs, including their ID, Directory, and Command line make command. The Submakes tab is available only if the job includes submakes.

Double-click an entry in the list to display the job's detail dialog.



## Make Details Dialog

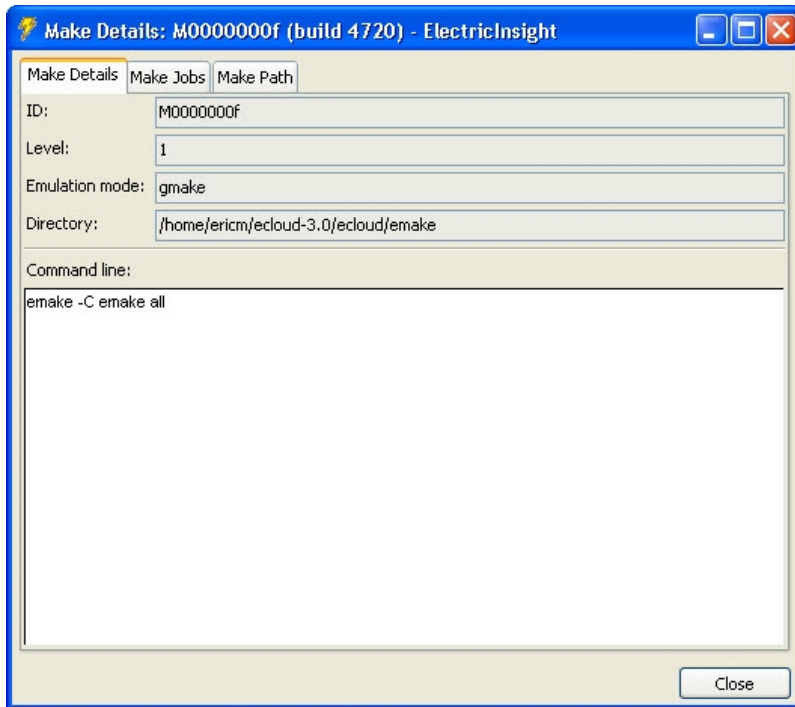
To open the Make Details dialog, double-click a make command, for example, in the Job Path tab of the Job Details dialog. Three tabs are available: Make Details, Make Jobs, and Make Path.

Topics:

- [Make Details](#)
- [Make Jobs](#)
- [Make Path](#)

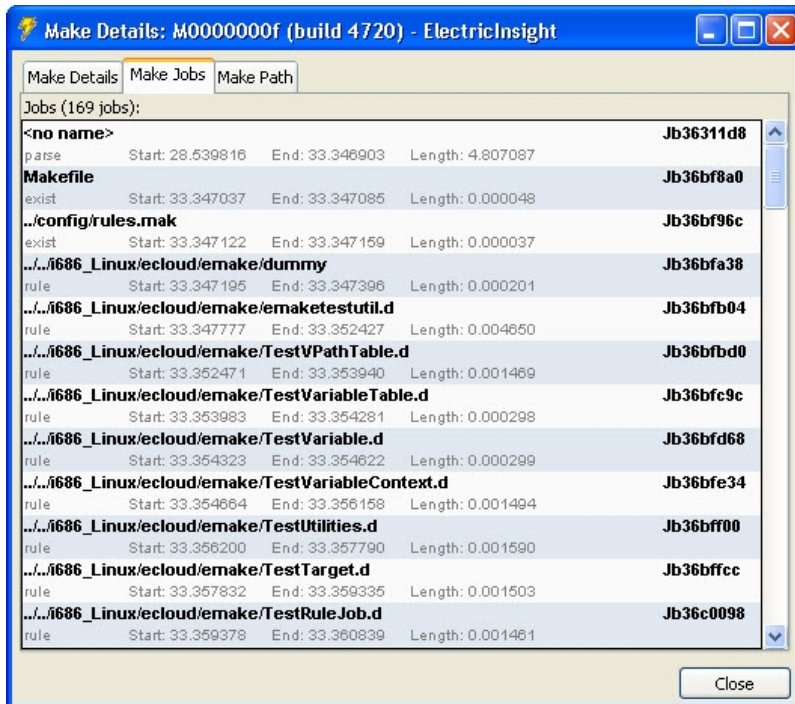
## Make Details

This tab displays information about make. The tab has the following fields: ID, Level, Emulation mode, Directory, and Command line.



## Make Jobs

This tab lists all jobs that the makefile runs.

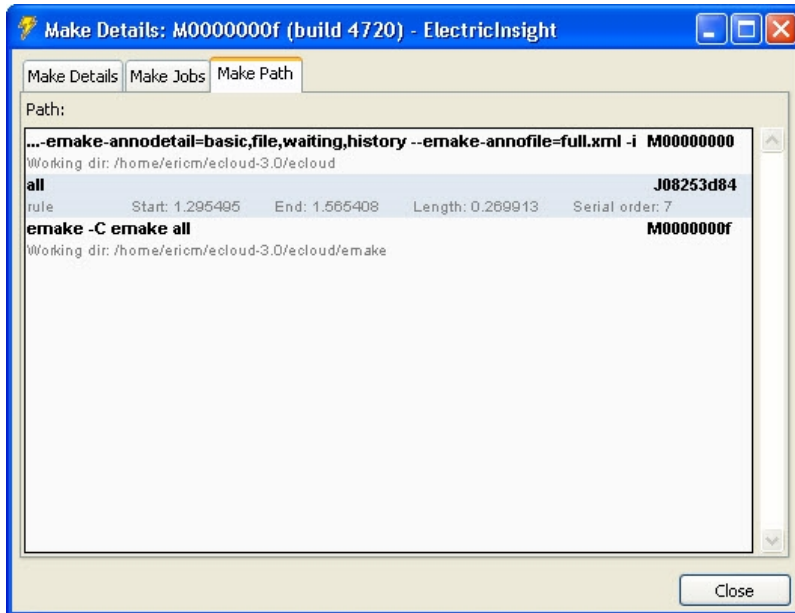




## Make Path

This tab displays the path to the makefile line(s) that created the make.

The first line contains the command that invokes the top level makefile. Typically, this line invokes another makefile. If it does, the second line displays the job ID. If it does not invoke another makefile, the second line displays the name of the target the job executes.



## Searching for Jobs

If you **know** the job ID you want to find:

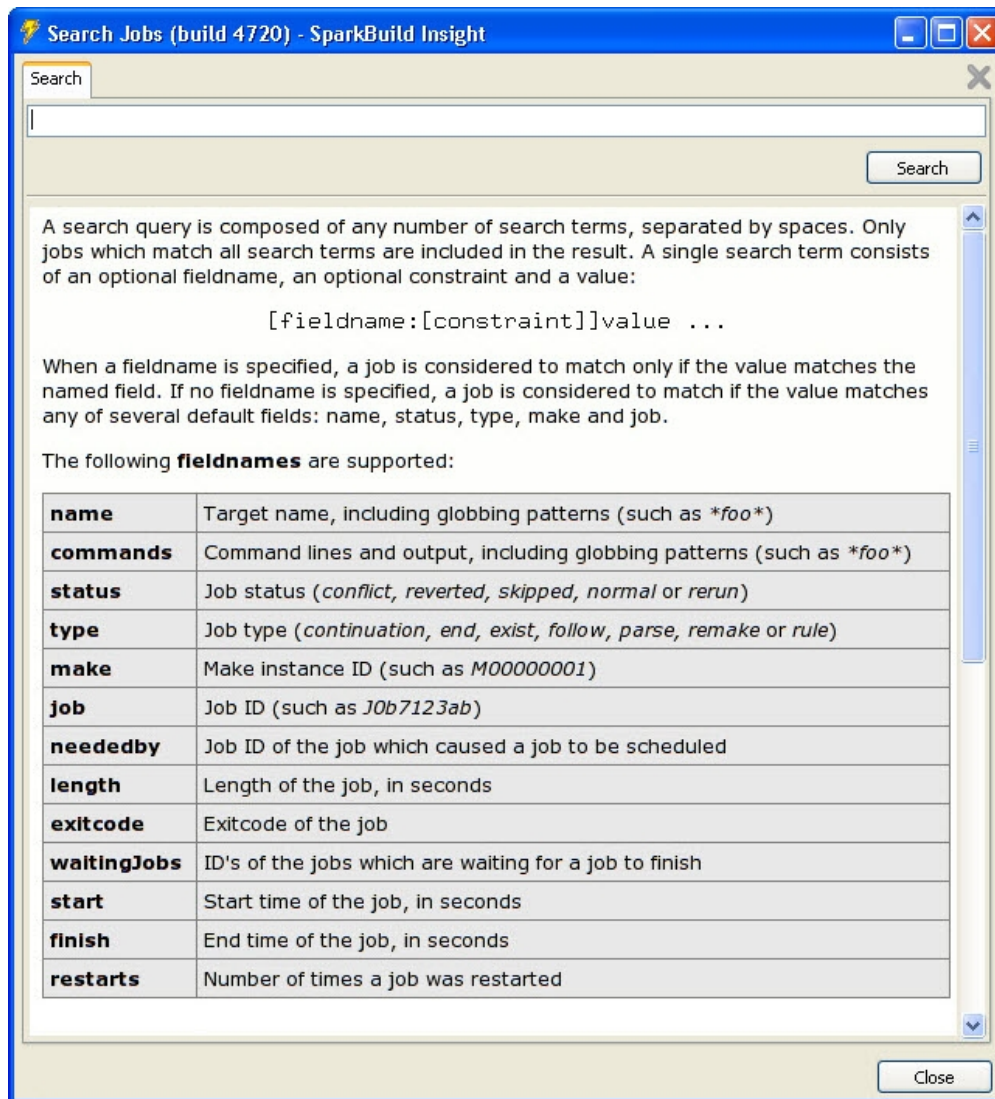
1. Go to Tools > Find job by ID to go to job details.



2. Enter the Job ID.
3. Click OK.

If you **do not know** the job ID, you can search for jobs using several criteria. Follow these steps:

1. Go to Tools > Search jobs.



2. Type-in the strings you want to find (example, "gcc conflict"). You can also constrain your search by searching for specific values in specific fields. See [Search ExamplesSupported Search Fields](#).
3. Click **Search**.
4. You can sort your results by selecting an option from the drop-down.
5. Double-click a row to display a job's Job Details dialog.
6. In the dialog's lower-right corner, click **Go to job**.

The job is now highlighted in the Agents & Jobs section.

## Search Examples

Finds rule jobs involved in conflicts:

```
rule conflict or type:rule status:conflict
```

Finds conflicts involving jobs named "export":

```
conflict name:export
```

Finds rerun jobs with names including “port”:

```
rerun name:*port*
```

Finds non-reverted jobs with names ending in “Makefile”:

```
!reverted name:*Makefile
```

## Supported Search Fields

The following fields are supported:

- name - the name of a job is the name of the target that created it. Continuation and parser jobs do not have names.
- commands - search makefile command lines and output. Globbing patterns are supported (for example, `commands:gcc *foo*`).
- status - possible values include:
  - normal - a successfully run job
  - conflict - a job involved in a conflict. Many of these jobs or jobs on which they rely are reverted. Typically, conflicts occur when there is no history file—when the environment does not provide Electric Make with enough information to predict all job dependencies.
  - rerun - some reverted jobs must be run with correct file system context. Rerun jobs are always serialized to guarantee correct context.
  - reverted - a job is reverted when Electric Make serializes job results and the job in question was executed before a (logically) preceding job that failed. The preceding job may have failed because of a conflict. Reverting a job restores the Agent’s file system to the state before the job was run.
  - skipped - skipped jobs are the same as reverted jobs except the skipped job did not run before the preceding job failed.
- type: possible values include:
  - continuation - a continuation job occurs when commands associated with a makefile target include a submake (also known as, *recursive* make). Typically, these submake commands begin with `$(MAKE)`. The commands below a submake, down to and including the next submake command (for the same target), comprise a continuation job. For a target, there are as many continuation jobs as there are submakes with commands after them.

```
all:
  echo abc
  $(MAKE) abc
  echo def
  echo ghi
  $(MAKE) ghi
  echo jkl
  $(MAKE) jkl
```

In this example, the first continuation job starts with `echo def` and ends with `$(MAKE) ghi`. The second and last continuation job consists of two commands involving `jkl`.

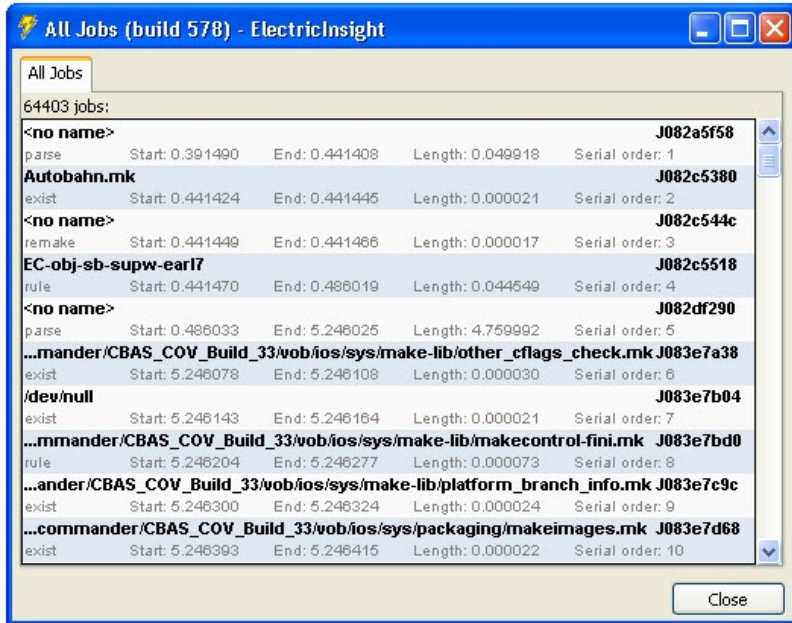
- end - clean-up tasks that occur at the end of makefiles (for example, removing intermediate files).
- exist - jobs corresponding to top-level targets without rules. Electric Make verifies the target exists.

- follow - one of these jobs follows every submake to handle submake output and exit status for the parent.
- parse - jobs that parse makefiles.
- remake - jobs that remake makefiles (only for GNU make).
- rule - lists jobs corresponding to makefile rules. Sorting results by name allows you to find the job that executed a specific rule. If you have several rules (in different makefiles) with the same name, you can display several Job Details dialogs at once.
- make - this is the ID of the makefile that created the job. The top-level makefile has ID 0 (zero). For example, `make:0` returns a list of jobs created directly from the top-level makefile.
- job - the job ID, for example, `job:J01b11200`
- neededby - the job in question is needed by another job. Specify the job ID of the job that needs the job in question, for example, `neededby:J01b11200`. This is useful for tracking inter-job dependencies.
- length - job length, in seconds. If you specify `length:12.34`, you find jobs that lasted exactly 12.34 seconds only. You can also search using `length:>12.34`, `length:>=12.34`, `length:==12.34`, `length:<12.34`, and `length:<=12.34`. To find jobs that lasted 10-15 seconds, specify `length:>=10.0 length:<=15.0`.
- exitcode - the job exit code. To find jobs with a non-zero exit code use `exitcode:!=0`. In general, use arithmetic relations (`=`, `>`, `<`, `>=`, `<=`) to search numerical values and use no relation symbols when searching for string values.
- waitingJobs:<jobID> - to find prerequisite jobs of <jobID>.
- start - job start time (in seconds) after the build start time. To find other jobs that started in the first minute of the job, use `start:<60`. To find jobs that took more than 60 seconds and started during the 10th minute into the build, type `length:>60 start:>=600 start:<660`.
- finish - the job end time (in seconds) after the build start time.
- restarts - lists jobs restarted a specified number of times. Jobs are restarted because of cluster sharing and agent failure.

**Note:** You can restrict the search scope for the following fields only: type, status, name, make, and job.

## Viewing All Jobs

If you want to list all of the build's jobs, go to Tools > View all jobs...



All Jobs (build 578) - ElectricInsight					
64403 jobs:					
<no name>	parse	Start: 0.391490	End: 0.441408	Length: 0.049918	Serial order: 1
Autobahn.mk	exist	Start: 0.441424	End: 0.441445	Length: 0.000021	Serial order: 2
<no name>	remake	Start: 0.441449	End: 0.441466	Length: 0.000017	Serial order: 3
EC-obj-sb-supw-earl7	rule	Start: 0.441470	End: 0.486019	Length: 0.044549	Serial order: 4
<no name>	parse	Start: 0.486033	End: 5.246025	Length: 4.759992	Serial order: 5
...mmander/CBAS_COV_Build_33/vob/ios/sys/make-lib/other_cflags_check.mk	exist	Start: 5.246078	End: 5.246108	Length: 0.000030	Serial order: 6
/dev/null	exist	Start: 5.246143	End: 5.246164	Length: 0.000021	Serial order: 7
...mmander/CBAS_COV_Build_33/vob/ios/sys/make-lib/makecontrol-fini.mk	rule	Start: 5.246204	End: 5.246277	Length: 0.000073	Serial order: 8
...ander/CBAS_COV_Build_33/vob/ios/sys/make-lib/platform_branch_info.mk	exist	Start: 5.246300	End: 5.246324	Length: 0.000024	Serial order: 9
...commander/CBAS_COV_Build_33/vob/ios/sys/packaging/makeimages.mk	exist	Start: 5.246393	End: 5.246415	Length: 0.000022	Serial order: 10

## Using the Hyperlog

The hyperlog presents an augmented version of the familiar build output log, leveraging the extra information provided by Electric Make to enable highlighting errors and warnings, folding of submake output, and line numbering.

To access the hyperlog, click the **Go to log** button on any Job Details dialog. When the hyperlog opens, you go directly to the corresponding job's location. You can also go to Tools > View build log... to access the hyperlog.

Double-clicking any line of log output displays the corresponding job's details dialog.

**Note:** The hyperlog is not available when monitoring a live build because the log file doesn't exist.

## Chapter 5: Reports

You can generate reports from the GUI and the command-line.

In the GUI, to display the Reports dialog, select Tools > Reports or click the reports icon on the toolbar. You can open one report at a time for each Insight instance, but you can open several Insight instances. You can run most reports by double-clicking the report name. Some reports have mandatory parameters that you must provide before running them.

Some reports listed here can be generated through both methods, and some reports can be generated through one method only.

With ElectricInsight's command-line reporting capability, the product can be integrated to generate batch trend reports as part of a build automation system. For a list of all command-line enabled reports and their associated commands, see [Command-Line Enabled Reports](#).

### Topics:

- [Build Manifest](#)
- [Build Metrics](#)
- [Build Summary](#)
- [Cluster Sharing](#)
- [Derived Files Analysis](#)
- [ElectricSimulator](#)
- [Export Timeline](#)
- [Files Modified Multiple Times](#)
- [Job Stats](#)
- [Job Time by Type](#)
- [Jobs by Agent](#)
- [Jobs by File](#)
- [Longest Jobs](#)
- [Longest Serial Chain](#)
- [Makefile Manifest](#)
- [Most Read Files](#)
- [Root Conflicts](#)
- [Serialization Analysis](#)
- [Creating a Custom Report](#)

## Build Manifest

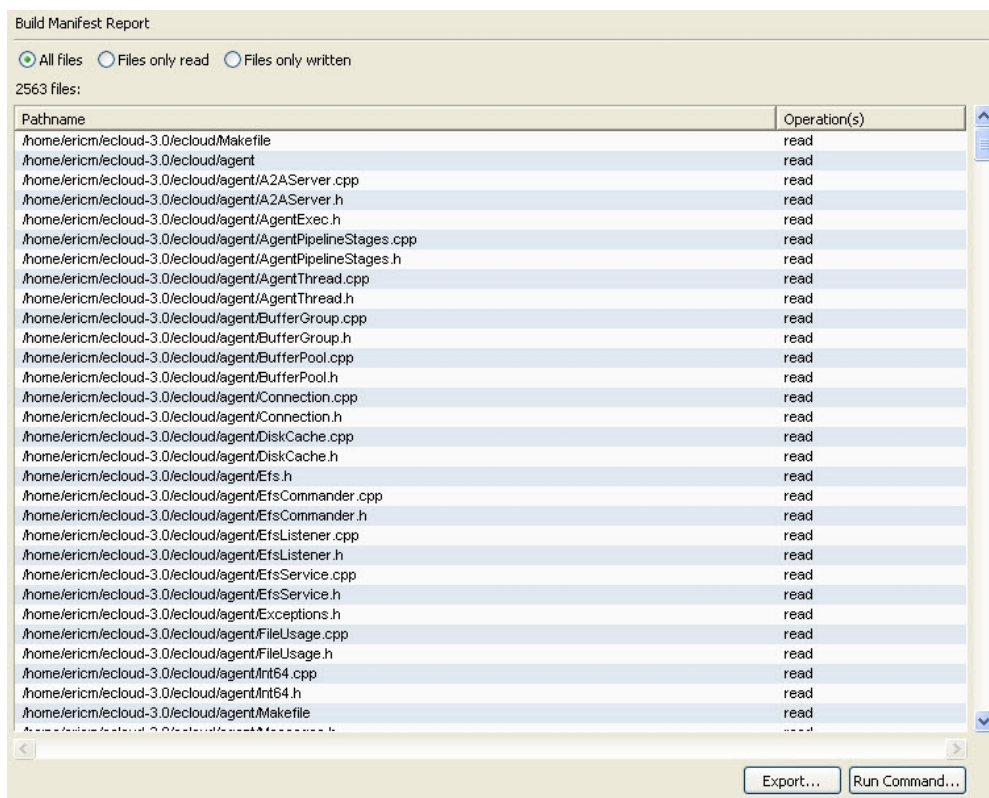
The Build Manifest report lists all files that were read and/or written by the build.

This report can help to:

- Verify that all files read by the build are the latest version from source control
- Add all files written by the build to an archive
- Produce a list of all inputs and a list of all outputs
- Identify files that are written but never read

This report requires

```
--emake-annodetail=basic, file
```



To export the report to a .csv file, click **Export**.

To run a shell command:

1. Click **Run Command**.
2. Type a command and a pathname in the Run Shell Command dialog.

The shell command is run with each file in the filtered list as an argument. Results and errors are displayed under their corresponding tabs.

## Build Metrics

This command-line-only report prints the content of the <metrics> element in the annotation.

Run:

```
einsight --report=BuildMetrics <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where *<annotation>* is the path and name of the annotation file.

Results format:

```
name,value
```

## Build Summary

This command-line-only report prints the information from the left-side of the main Insight display.

Run:

```
einsight --report=BuildSummary <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where *<annotation>* is the path and name of the annotation file.

Results format:

```
name,value
```

## Cluster Sharing

The Cluster Sharing report shows the total work time lost (sum of aborted job run-times) and net build time lost (total work time minus overlaps) due to cluster sharing over the lifetime of the build.

This report is useful if your build aborts jobs because an agent host is de-allocated from the build. The host may have been allocated to another build, moved to another cluster, or simply shut down. The aborted jobs must be rerun, resulting in a slower build.

This report does not consider the effect of not using some agent hosts because other builds are using them. To evaluate this factor, use the [ElectricSimulator](#) report.

### Example

- Job A starts 5 seconds into the build.
- Job B starts 10 seconds into the build.
- Both jobs are terminated 20 seconds into the build.

The total work time lost is 15 + 10 seconds, but the net build time lost is only 15 seconds because the build time lost by B overlaps the build time lost by A.



37.80s total work time lost due to cluster sharing;  
37.80s net build time lost.

Job ID	Length	Restarts	Wasted time	Name
J41e36050	12.69	2	37.8	c:\builds\acme\xyz\998874930_abc\productcode\giga\buildfolders\mytargets\_45356ghi\m

## Derived Files Analysis

The Derived Files Analysis report allows you to view a specified file's affected outputs. For example, this report answers the question, "If I change file X, what outputs will be affected?" A file is likely to be affected if it is produced concurrently with, or subsequent to, reads of the specified file.

This report requires

```
--emake-annodetail=file,history
```

Derived Files Report (5 operations)

Enter input file(s), one per line:

/home/ericm/eccloud-3.0/eccloud/compressor/minilzo.h

Clear Run

The following files were changed when the input files changed:

Filename	Full Path
emake	/home/ericm/eccloud-3.0/i686_Linux/eccloud/emake/emake
agent	/home/ericm/eccloud-3.0/i686_Linux/eccloud/agent/agent
minilzo.o	/home/ericm/eccloud-3.0/i686_Linux/eccloud/compressor/minilzo.o

Got 3 steps for file /home/ericm/eccloud-3.0/i686\_Linux/eccloud/agent/agent

Immediate Precursor File	Resultant File	Oper.
/home/ericm/eccloud-3.0/i686_Linux/eccloud/compressor/compressor.a	/home/ericm/eccloud-3.0/i686_Linux/eccloud/agent/agent	creat
/home/ericm/eccloud-3.0/i686_Linux/eccloud/compressor/Compressor.o	/home/ericm/eccloud-3.0/i686_Linux/eccloud/compressor/compressor.a	creat
/home/ericm/eccloud-3.0/eccloud/compressor/minilzo.h	/home/ericm/eccloud-3.0/i686_Linux/eccloud/compressor/Compressor.o	creat

Copy Report

To run the report:

1. Enter the file(s) with full paths in the input field.
2. Click **Run**.

The middle display area lists files that will change when the input file changes. Double-click a file in this list to display its chain of operations in the lower display area.

Double-click an item in the lower display area to display its Job Details dialog.

To clear all fields, click **Clear**.

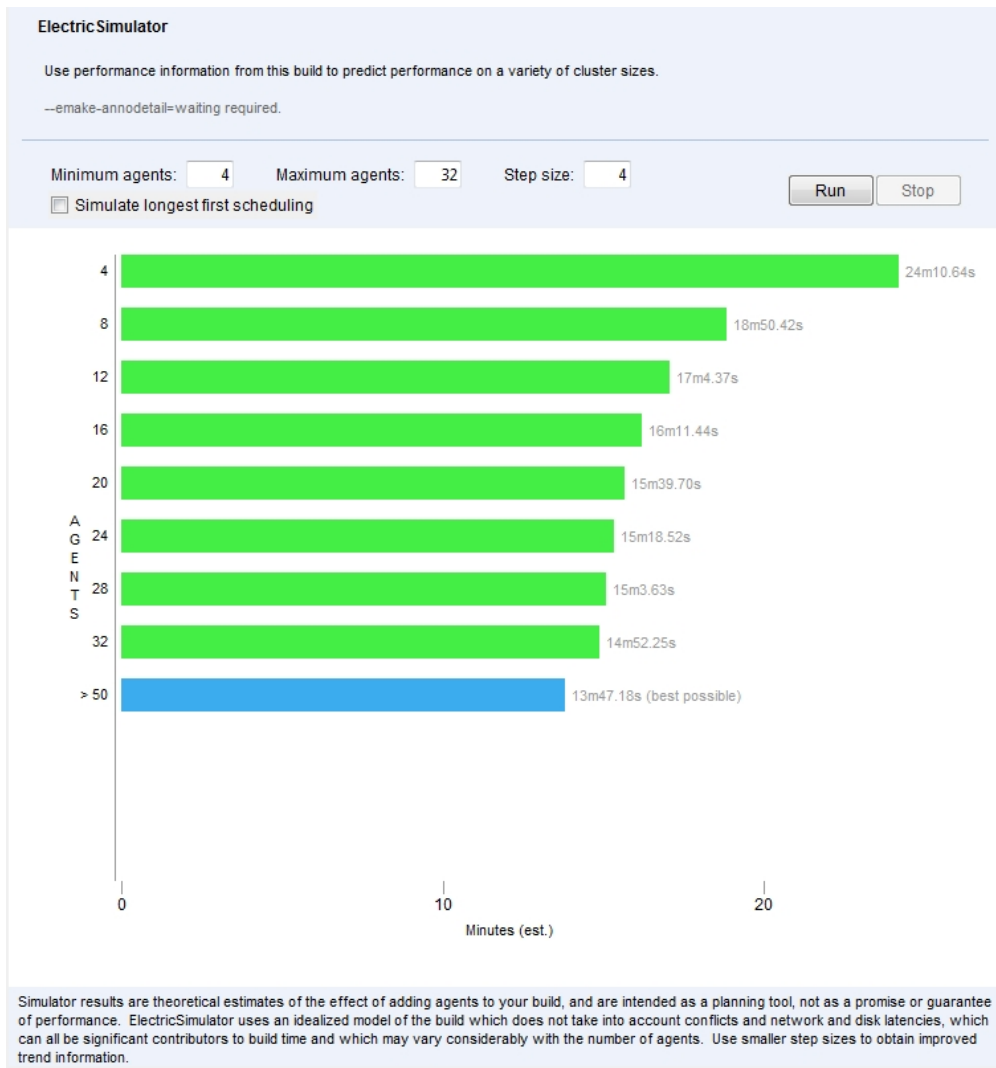
To copy the displayed chain of operations (in the lower display area) to the clipboard, click **Copy Report**.

## ElectricSimulator

The ElectricSimulator report predicts build performance on a variety of cluster sizes. The last time indicator (blue bar) is the best possible build time.

This report requires

```
--emake-annodetail=waiting
```



**Note:** The simulator time results are estimates only. They do not indicate guaranteed performance.

- **Minimum agents** - the minimum number of agents to simulate. Default is 4.
- **Maximum agents** - the maximum number of agents to simulate. Default is 32.
- **Step size** - the number of agents to increment the count each time. Default is 4.

### Command-Line Interface Report

Run this command:

```
einsight --report="ElectricSimulator [minagents] [maxagents] [stepsize]" <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where:

*[minagents]* (optional) is the minimum number of agents to simulate. Default is 4.

*[maxagents]* (optional) is the maximum number of agents to simulate. Default is 32.

*[stepsize]* (optional) is the number of agents to increment the count each time. Default is 4.

`<annotation>` is the path and name of the annotation file.

Results format:

```
agents,duration
```

## Export Timeline

This command-line-only report exports the main build timeline view to a graphic.

Run:

```
einsight --report=ExportTimeline <filename> [mode] [scale] [rowheight] [rowspace] <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where:

- `<filename>` is the name of the PNG file to write to. (Required)
- `[mode]` indicates if you want the image in color or black and white. Valid values are `color` or `bw`. Default is `color`. (Optional)
- `[scale]` is the number of seconds per pixel. Default is `2`. (Optional)
- `[rowheight]` is the number of pixels per row of jobs. Default is `1`. (Optional)
- `[rowspace]` is the number of pixels separating rows. Default is `1`. (Optional)
- `<annotation>` is the path and name of the annotation file. (Required)

## Files Modified Multiple Times

This report shows which files are modified (created, updated, or deleted) multiple times during a build.

Jobs involving these files require careful serialization to ensure the file operations sequence is performed in the correct order.

This report requires

```
--emake-annodetail=file
```

The following files were modified more than once: Filter: \*

Writes	Filename	Location
4	tile0.6	/home/ericm/eccloud-3.0/686_Linux/eccloud/buildplotter/insight.vfs/lib
3	allTheme.tcl	/home/ericm/eccloud-3.0/686_Linux/eccloud/installer/eccloud/ecadduser.vfs/lib/tile0.6
3	tdom.tcl	/home/ericm/eccloud-3.0/686_Linux/eccloud/buildplotter/insight.vfs/lib/tdom0.8.0
3	ncgi	/home/ericm/eccloud-3.0/686_Linux/eccloud/annolib/anno2log.vfs/lib/tcllib1.6
3	mwutil.tcl	/home/ericm/eccloud-3.0/686_Linux/eccloud/buildplotter/insight.vfs/lib/tablelist4.0/scripts
3	pkgIndex.tcl	/home/ericm/eccloud-3.0/686_Linux/eccloud/buildplotter/insight.vfs/lib/cursor
3	tile.tcl	/home/ericm/eccloud-3.0/686_Linux/eccloud/buildplotter/insight.vfs/lib/tile0.6

rule	Start: 191.148324 End: 193.202772 Length: 2.054448	create	Jb36be640
rule	Start: 200.399859 End: 204.517484 Length: 4.117625	modifyAtts	Jb36be7d8
rule	Start: 200.399859 End: 204.517484 Length: 4.117625	read	Jb36be7d8
rule	Start: 200.399859 End: 204.517484 Length: 4.117625	unlink	Jb36be7d8

After you run the report, filter the results:

1. Type the string you want to filter for in the **Filter** field.

Use an asterisk to match any number of characters, and use a question mark to match any single character.

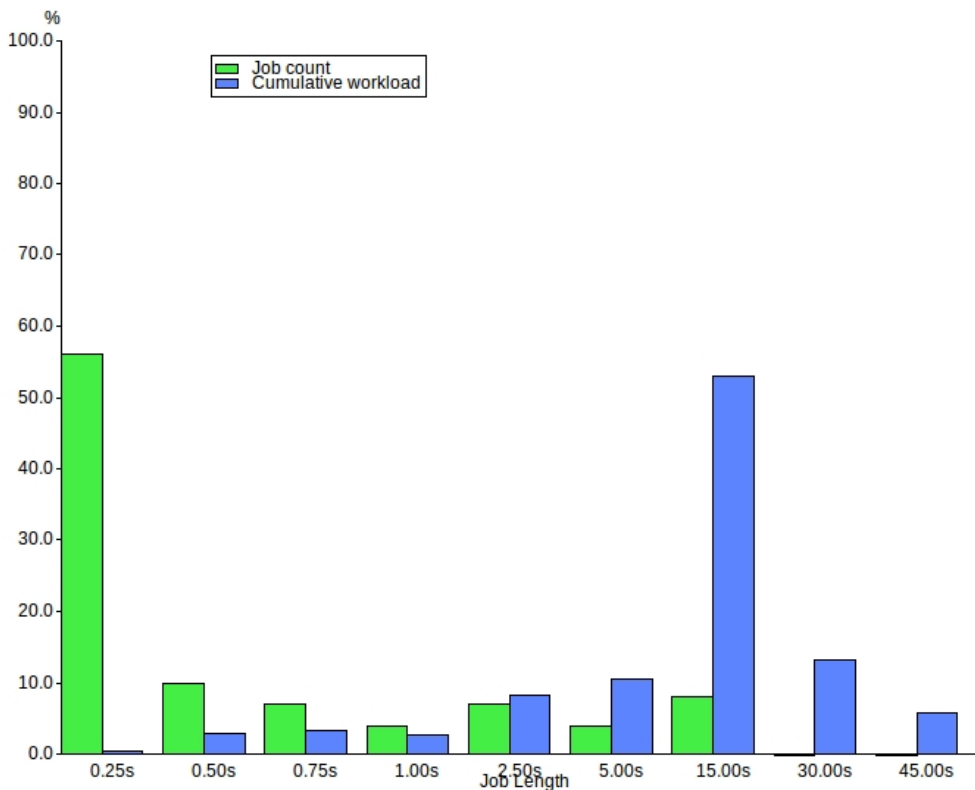
You can also use simple regular expressions, for example, `*[xz].o` and `*[x-z].o`

Filters are case sensitive.

2. Press **Enter**.

## Job Stats

This report groups jobs based on duration and shows the percentage of jobs in each group, as well as the percentage of total build workload represented by the jobs in each group.



### Command-Line Interface Report

Run this command:

```
einsight --report=JobStats <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where *<annotation>* is the path and name of the annotation file.

You can also limit results to a job class, such as "compile" or "link". To limit your results, run this command:

```
einsight --report="JobStats <jobclass>" <annotation>
```

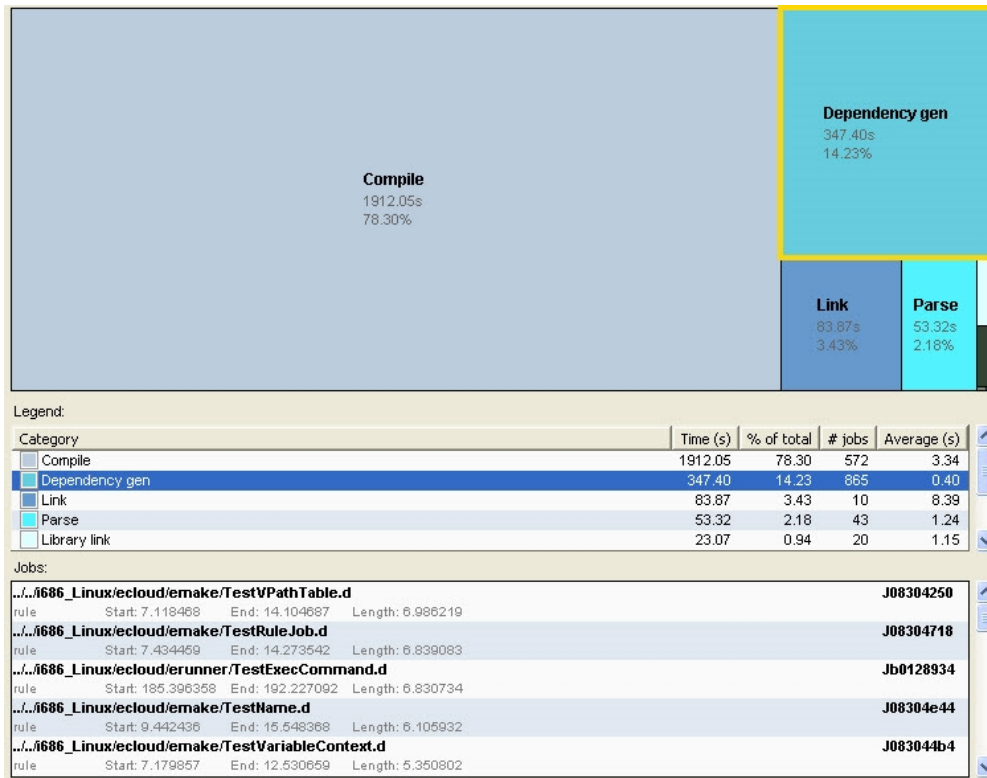
Where *<jobclass>* is the only job class you want to see, which can be link, compile, parse, etc.

Results format:

```
duration,count,time,count percent,time percent
```

## Job Time by Type

This report shows the portion of total job time consumed by each job type. The area covered by each color is proportional to the total time consumed by its corresponding job type.



The legend displays corresponding colors for job types. The following information is available:

- Category
- Time (s)
- % of total
- # jobs
- Average (s)

**Note:** To sort data, click a column heading.

Clicking a row outlines the corresponding area in the treemap in yellow and lists all jobs of that type in the Jobs section. Double-clicking a job from the list displays that job's details. You can also display all jobs of a specific type by clicking an area of the treemap.

## Command-Line Interface Report

Run this command:

```
einsight --report=JobTimeByType <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where `<annotation>` is the path and name of the annotation file.

Results format:

```
class,count,percent,seconds,average
```

## Jobs by Agent

This report lists the number of jobs run by each agent in the build.

Jobs by agent:

Agent	Jobs on this agent	Jobs duration
LocalNode	4	0.05s
linbuild1-1	93	3m29.23s
linbuild1-2	133	3m25.84s
linbuild2-1	238	3m19.58s
linbuild2-2	162	3m22.46s
linbuild3-1	134	3m24.86s
linbuild3-2	156	3m28.21s
linbuild4-1	116	3m23.04s
linbuild4-2	209	3m20.71s
linbuild5-1	74	3m20.93s
linbuild5-2	184	3m19.18s
linbuild6-1	109	3m22.24s
linbuild6-2	90	3m25.58s

<no name>				J082679b8
parse	Start: 1.584745	End: 1.811608	Length: 0.226863	
Makefile				J0824a7d8
exist	Start: 1.811663	End: 1.811704	Length: 0.000041	
../config/rules.mak				J0824a8a4
exist	Start: 1.811721	End: 1.811755	Length: 0.000034	
.././686_Linux/ecloud/testmain/dummy				J0824a970
rule	Start: 1.811771	End: 1.831088	Length: 0.019317	
.././686_Linux/ecloud/testmain/TestTester.d				J0824aa3c
rule	Start: 1.831125	End: 3.285789	Length: 1.454664	
.././686_Linux/ecloud/Util/TestTime.d				Jb421461c
rule	Start: 3.285834	End: 3.823118	Length: 0.537284	
.././686_Linux/ecloud/Util/TestStringPool.d				Jb4214e14
rule	Start: 3.823151	End: 6.698036	Length: 2.874885	
.././686_Linux/ecloud/Util/Exec.d				Jb4217c4c
rule	Start: 6.708147	End: 7.120006	Length: 0.411859	
.././686_Linux/ecloud/Util/Pipeline.o				Jb421fa38
rule	Start: 12.640807	End: 19.538568	Length: 6.897761	
.././686_Linux/ecloud/Util/System-unix.o				Jb4220c8c
rule	Start: 19.538635	End: 23.154224	Length: 3.615589	

Click an agent to display the jobs that it ran. Double-click a job to display its details.

## Jobs by File

This report lists which jobs read or wrote a particular file.

Type in or browse to the file you want to analyze and click **Analyze**.

## Longest Jobs

This report lists the build's 10 longest jobs.



Longest jobs:			
../i686_Linux/ecldoud/emade/emade			Jb36c7c1c
rule	Start: 114.173690	End: 156.010421	Length: 41.836731
../i686_Linux/ecldoud/webui/mod_ecldoud/mod_ecldoud_comment_update.o			Jb02b9168
rule	Start: 118.820799	End: 160.133297	Length: 41.312498
<no name>			J0826b560
parse	Start: 2.053661	End: 32.279670	Length: 30.226009
../i686_Linux/ecldoud/emade/GMake.o			Jb36c79b8
rule	Start: 84.076576	End: 114.173629	Length: 30.097053
../i686_Linux/ecldoud/emade/Build.o			Jb36c5048
rule	Start: 33.639376	End: 61.030681	Length: 27.391305
../i686_Linux/ecldoud/emade/ShellEmulator.o			Jb36c70f4
rule	Start: 72.897960	End: 95.217902	Length: 22.319942
../i686_Linux/ecldoud/emade/RemoteNode.o			Jb36c6dc4
rule	Start: 67.761896	End: 89.233741	Length: 21.471845
../i686_Linux/ecldoud/agent/Session.o			J086bc1f8
rule	Start: 171.062765	End: 191.910578	Length: 20.847813
../i686_Linux/ecldoud/emade/RelocTable.o			Jb36c6b60
rule	Start: 63.667325	End: 84.076530	Length: 20.409205
../i686_Linux/ecldoud/emade/History.o			Jb36c5dd4
rule	Start: 47.863931	End: 67.761811	Length: 19.897880

To view the Job Details dialog, double-click a job ID.

## Command-Line Interface Report

Run this command:

```
einsight --report=LongestJobs <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where *<annotation>* is the path and name of the annotation file.

Results format:

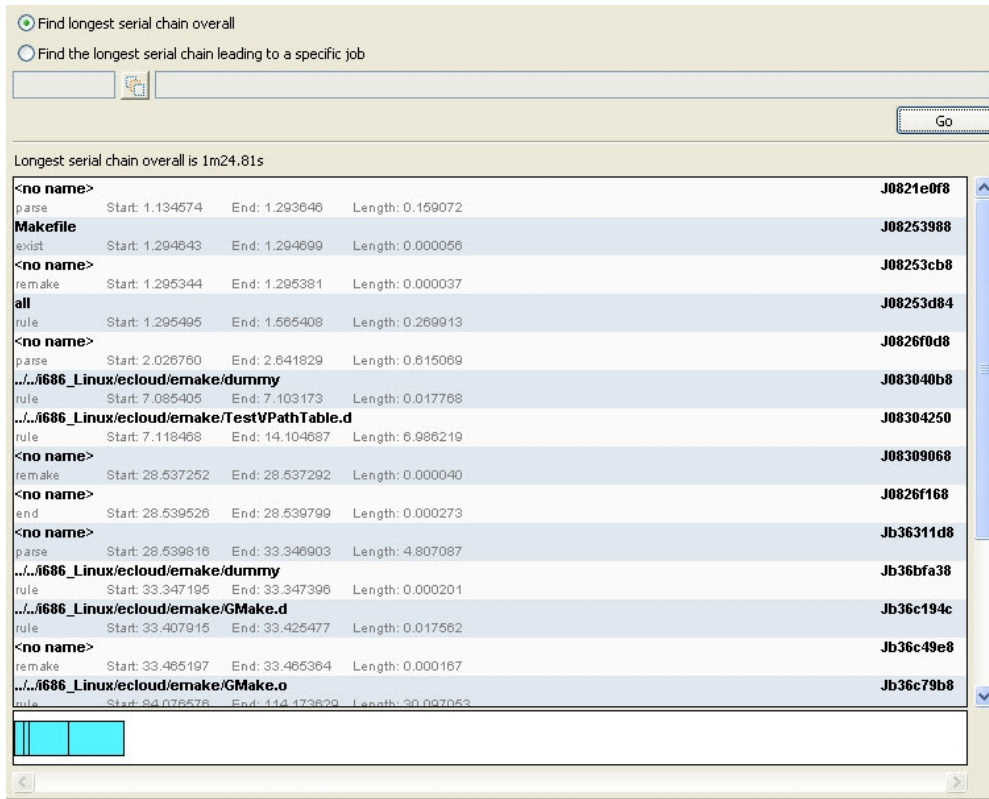
```
job id,duration,name
```

## Longest Serial Chain

This report displays the sequence of serialized jobs with the longest end-to-end runtime in the build. The longest serial chain represents a lower bound on the build runtime. Without changing the structure or content of the build, the runtime cannot be less than the longest serial chain.

This report requires

```
--emade-annodetail=waiting,history
```



You can display the longest serial chain overall or the longest serial chain leading to a specific job.

To display the Job Details dialog, double-click a job.

## Command-Line Interface Report

Run this command:

```
einsight --report="LongestSerialChain [job id]" <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where:

`[job id]` is a job ID (optional). Specifying a job ID means that you want to find the longest serial chain leading to that job. If you don't specify a job ID, the report returns the longest serial chain overall.

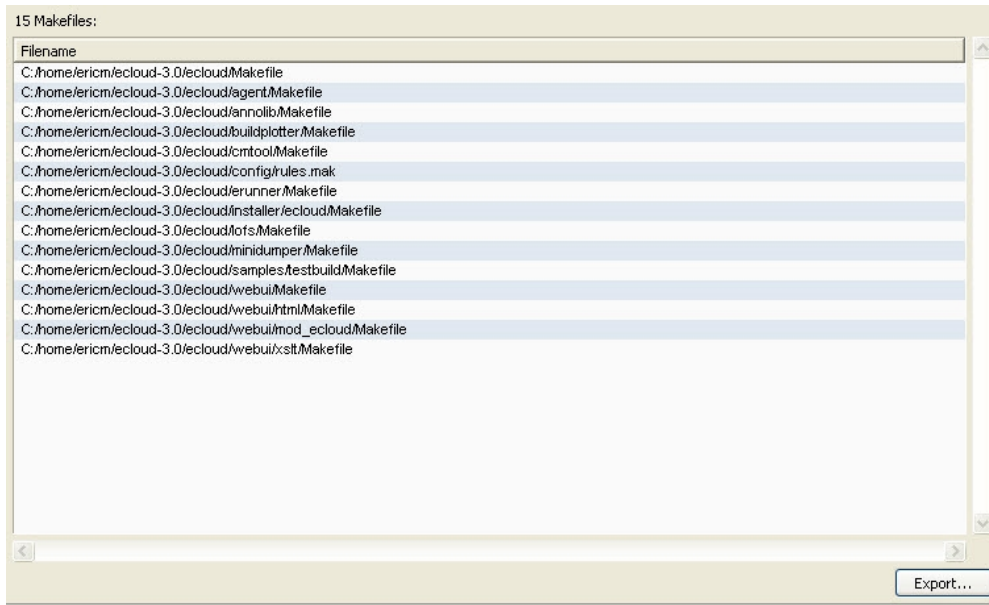
`<annotation>` is the path and name of the annotation file.

Results format:

```
job,type,length,name
```

## Makefile Manifest

This report lists all makefiles used in the build.



To export the list to a file, click **Export**.

## Most Read Files

This report lists the number of times a file was read, its name, type, and location.

Most Read Files:

Reads	Filename	Type	Location
75	ims_ic.h	file	X:/product/ATN/app/ada/ims/core/include
74	drv_bits_def.h	file	X:/product/ATN/app/drv/drvarch/platform/include
74	xml_pub.h	file	X:/vrp/util/xml
74	xml_def.h	file	X:/vrp/util/xml
74	ac_manage.h	file	X:/product/ATN/app/ada/ims/shell/include
74	xml_type.h	file	X:/vrp/util/xml
74	drv_clk_pub.h	file	X:/product/ATN/app/include/drv
74	xml_func.h	file	X:/vrp/util/xml
74	drv_clk_intf.h	file	X:/product/ATN/app/drv/clkdrv
74	gfpi_ims_pub.h	file	X:/product/ATN/app/include/gfpi
74	ims_shadow.h	file	X:/product/ATN/app/ada/ims/core/include
73	itnim_inc_common.h	file	X:/vrp/pos/software/common_level2_to_others/common_to_pdt/tnlm
73	srv_pub.h	file	X:/product/ATN/app/ada/security/mse
73	l3vpn_inc.h	file	X:/vrp/pos/software/common_level2_to_others/common_to_pdt/l3vpn
73	fsdl_pub.h	file	X:/product/ATN/app/include/ada
73	srv_qosinfo.h	file	X:/product/ATN/app/ada/security/mse
72	gfpi_ims_acm.h	file	X:/product/ATN/app/dvra/gfpi/ims/include
72	gfpi_ims_def.h	file	X:/product/ATN/app/dvra/gfpi/ims/include

## Command-Line Interface Report

Run this command:

```
einsight --report=MostReadFiles <annotation>
```

**Note:** On Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

Where *<annotation>* is the path and name of the annotation file.

Results format:

```
count,type,name,location
```

## Root Conflicts

This report lists all root conflicts in the build.

Insight divides all conflicts found in a build into two categories:

- root conflicts, which are conflicts that are not caused by an earlier conflict
- conflicts resulting from earlier conflicts (for example, jobs that are in conflict with jobs that are rerun jobs)

Root conflicts (619 jobs):				
<b>net2890klib</b>	rule	Start: 70.075000	End: 71.418100	Length: 1.343100
<b>rne_mdd</b>	rule	Start: 70.075000	End: 71.425000	Length: 1.350000
<b>3c90xdbg</b>	rule	Start: 70.448800	End: 71.008900	Length: 0.560100
<b>eboot</b>	rule	Start: 70.448800	End: 71.489500	Length: 1.040700
<b>iplcommon</b>	rule	Start: 70.448900	End: 71.327000	Length: 0.878100
<b>blcommon</b>	rule	Start: 70.845500	End: 71.382200	Length: 0.536700
<b>bootpart</b>	rule	Start: 70.845500	End: 71.534300	Length: 0.688800
<b>celog</b>	rule	Start: 70.846100	End: 71.763600	Length: 0.917500
<b>PCIreg</b>	rule	Start: 71.341400	End: 72.131400	Length: 0.790000
<b>cecap</b>	rule	Start: 71.341400	End: 72.092900	Length: 0.741500
<b>fal</b>	rule	Start: 71.341400	End: 72.116500	Length: 0.775100
<b>ecclib</b>	rule	Start: 71.609000	End: 72.448700	Length: 0.839700
<b>sdnpcik</b>	rule	Start: 71.609100	End: 72.590500	Length: 0.971400
<b>stratak</b>	rule	Start: 71.609100	End: 72.399400	Length: 0.790300
<b>faslk</b>	rule	Start: 71.609100	End: 72.677300	Length: 1.068200

## Serialization Analysis

This report details dependencies between a pair of jobs. For example, you see in the Annotation tab that a job has waitingJobs and you want to know why the jobs are waiting.

In some cases you can use this report to understand the longest serial chain, but dependencies may exist that are not within the scope of the Serialization analysis report. For example, it is common for a rule job to be preceded by a parse job. This type of logical dependency is not included in the Serialization analysis report.

This report requires

```
--emake-annodetail=waiting,history
```

but works best with

```
--emake-annodetail=waiting,history,registry
```

Specify two jobs for serialization analysis:

Jb3c984e4	/y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/APICreateRepo
Jb3ca1720	/y/sb/livbuilds_int_87_870_rebuildsc7999_M/vobs/all/api_v2/api_v2

---

**Explicit dependency**

These jobs are serialized by the following dependency:

Makefile: /y/sb/livbuilds\_int\_87\_870\_rebuildsc7999\_M/vobs/all/api\_v2/.Makefile.e

Dependency: /y/sb/livbuilds\_int\_87\_870\_rebuildsc7999\_M/vobs/all/api\_v2/api\_v2 : /y/sb/live-000

---

**File-level dependencies**

The following files are modified by Jb3c984e4 and used by Jb3ca1720:

	First	Second
/94_C/vobs/sub_code2/ppapi_v2/APICreateRepoProfile_api_vobsfile_v3_1.o	create	read

1. Type the ID of the job you started first in the first job field.
2. Type the ID of the waiting job in the second job field.
3. Click **Locate Dependency**.

**Note:** If you know where the jobs [of interest] appear in the main jobs display, you can use the buttons next to the job fields to select the jobs.

## Command-Line Enabled Reports

This section lists all command-line enabled reports and their associated commands.

For all reports, *<annotation>* is required and represents the path and name of the annotation file.

**IMPORTANT:** For Windows, use `einsight-cmd` instead of `einsight` when running reports from the command-line.

### Build Metrics

```
einsight --report=BuildMetrics <annotation>
```

Results format:

```
name,value
```

### Build Summary

```
einsight --report=BuildSummary <annotation>
```

Results format:

```
name,value
```

## ElectricSimulator

```
einsight --report="ElectricSimulator [minagents] [maxagents] [stepsize]" <annotation>
```

Where:

*[minagents]* (optional) is the minimum number of agents to simulate. Default is 4.

*[maxagents]* (optional) is the maximum number of agents to simulate. Default is 32.

*[stepsize]* (optional) is the number of agents to increment the count each time. Default is 4.

Results format:

```
agents,duration
```

## Export Timeline

```
einsight --report=ExportTimeline <filename> [mode] [scale] [rowheight] [rowspace] <annotation>
```

Where:

- *<filename>* is the name of the PNG file to write to. (Required)
- *[mode]* indicates if you want the image in color or black and white. Valid values are `color` or `bw`. Default is `color`. (Optional)
- *[scale]* is the number of seconds per pixel. Default is 2. (Optional)
- *[rowheight]* is the number of pixels per row of jobs. Default is 1. (Optional)
- *[rowspace]* is the number of pixels separating rows. Default is 1. (Optional)

## Job Stats

```
einsight --report=JobStats <annotation>
```

You can also limit results to a job class, such as "compile" or "link". To limit your results, run this command:

```
einsight --report="JobStats <jobclass>" <annotation>
```

Where *<jobclass>* is the only job class you want to see, which can be link, compile, parse, etc.

Results format:

```
duration,count,time,count percent,time percent
```

## Job Time by Type

```
einsight --report=JobTimeByType <annotation>
```

Results format:

```
class,count,percent,seconds,average
```

## Longest Jobs

```
einsight --report=LongestJobs <annotation>
```

Results format:

```
job id,duration,name
```

## Longest Serial Chain

```
einsight --report="LongestSerialChain [job id]" <annotation>
```

Where *[job id]* is a job ID (optional). Specifying a job ID means that you want to find the longest serial chain leading to that job. If you don't specify a job ID, the report returns the longest serial chain overall.

Results format:

job,type,length,name

### Most Read Files

```
einsight --report=MostReadFiles <annotation>
```

Results format:

count,type,name,location

## Creating a Custom Report

You can create custom reports and make them accessible in Insight.

A report typically consists of:

- a single .tcl source file containing Tcl code required to perform build annotation analysis
- the Tk code required to display the result in the Insight UI

An example custom report could be one that displays the number of jobs run by each agent used in the build.

Follow these steps to create a custom report:

1. Create a Tcl script that uses `CreateReport` and `ConfigureReport` to declare and set attributes for a report, using this format:

```
CreateReport name ?-option value ...?
```

Valid options:

-command	Command for generating a text-only version of the report
-guicommand	Command for generating a GUI version of the report
-desc	Description of the report
-requires	Space-separated list of anno detail levels required for the report
-cleanup	Command to invoke to clean up state associated with the report, e.g., when a new anno file is loaded

You *must specify at least one of* `-command` and `-guicommand`.

For example:

```
set report [CreateReport "MyCustomReport"]
ConfigureReport $report -desc "The report description \
a continuation of the report description."
ConfigureReport $report -requires {basic file waiting history}
ConfigureReport $report -command RunCustomReport
ConfigureReport $report -uicommand MakeCustomReportUI
```

**Notes:**

- Make sure the console version of the report returns the report results as a text string (i.e., don't just print directly with [puts]).
  - If you're running the Insight GUI and you invoke a custom report that has no `-guicommand`, Insight will run the `-command` version, capture the output, and display it as raw text in the GUI.
2. Add to the Tcl script the function to perform the analysis and display results. The function is invoked with two arguments:
    - The name of the “widget” the function needs to create to display results.
    - (The `progressVar` argument is optional.) The name of the variable the function needs to update with progress information, from 0.0 to 1.0. This information controls the “progress bar” when the report is generated.

```
proc CreateJobsByAgentReport {w progressVar} {
```

In addition to function arguments, a global variable `anno` is available for reports. This is a handle for the `annolib` object containing build annotation information currently loaded in ElectricInsight. To access this information in your report, import the `anno` variable: `global anno`

3. Add Tcl code to perform the analysis on the annotation information:

```
# Iterate through all the jobs in the build, counting
# the number of jobs run by each agent.

array set count {}
$anno jobiterbegin
while { [$anno jobitermore] } {
    set job [$anno jobiternext]
    set agent [$anno job agent $job]
    if { $agent ne "" } {
        if { [info exists count($agent)] } {
            incr count($agent)
        } else {
            set count($agent) 1
        }
    }
}
```

4. Write Tk code to display results.

Insight includes the `[Tile]` widgets and the `[Tablelist]` widget. For this report, a simple tablelist can display the results:

```
# The “count” array contains the number of jobs run by
# each agent.
# We create a simple tablelist (multi-column listbox)
# to display the results.

frame $w
ttk::label $w.label -text “Jobs by agent:” -anchor w
tablelist::tablelist $w.results -columns {
    0 “Agent”
    0 “Jobs run by this agent”
} -height 10 -width 80 -borderwidth 1 -stretch end \
  -font TkDefaultFont -background gray98 \
  -stripebackground \#e0e8f0 \
  -labelcommand tablelist::sortByColumn
$w.results columnconfigure 1 -sortmode integer
grid $w.label -sticky ew
```



```
grid $w.results -sticky nsew
grid columnconfigure $w 0 -weight 1
grid rowconfigure $w 1 -weight 1

foreach agent [lsort [array names count]] {
    $w.results insert end [list $agent $count($agent)]
}
return $w
}
# end of procedure CreateJobsByAgentReport
```

5. Save the .tcl files that define your new report in one of the following directories.

- On Linux:
  - /opt/eccloud/ElectricInsight/reports
  - \$HOME/.eccloud/ElectricInsight/reports
- On Windows:
  - c:\eccloud\ElectricInsight\reports
  - \$USERPROFILE\Electric Cloud\ElectricInsight\reports

On startup, Insight scans the directories listed above for .tcl files defining new reports and automatically includes the reports the next time Insight starts. Reports saved in the Electric Cloud directories listed above are available to all users running Insight. Reports in other locations are available to a single user only.

6. Select Tools > Reload reports.

## Chapter 6: Understanding Build Performance

This section provides guidance for using Insight to help you understand build performance issues.

- Specify the right annotation detail level

At a minimum, specify `annodetail` to `basic`. Additional detail levels enable more sophisticated analysis of the build. For maximum performance analysis, set `annodetail` to `"basic,lookup,waiting,history,registry"`. Because additional information may affect performance slightly, specify basic detail level only, unless you are actively pursuing a performance issue.

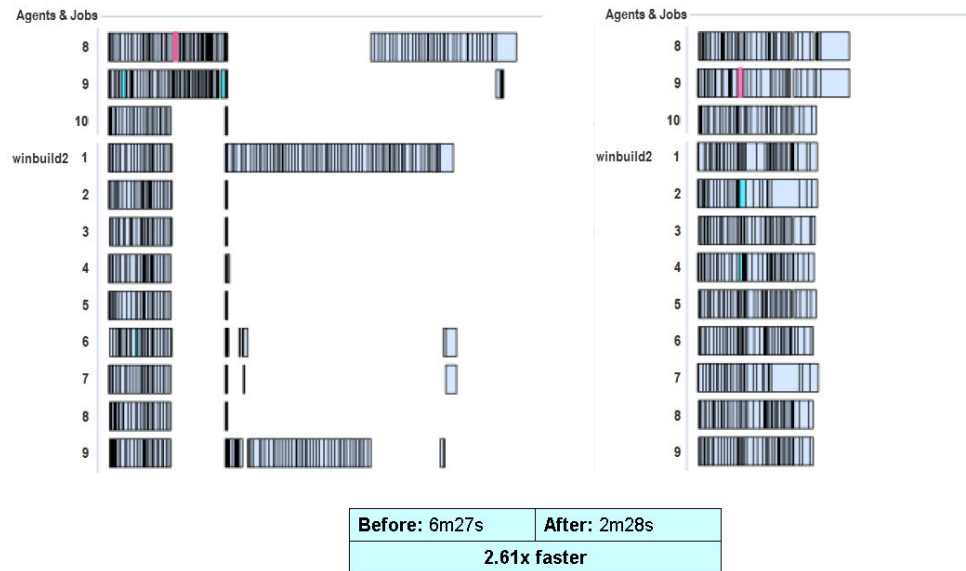
- Use job details

When viewing an annotation file in Insight, you can see if your build is slow due to excessive conflicts, over serializations, or insufficient decomposition of build steps. You can then drill-down into job details by double-clicking individual jobs in the Agents & Jobs section.

- Use reports

- The Longest serial chain report can help estimate the best possible performance you can reasonably get from the build.
- The Serialization analysis report can help you understand why jobs are serialized. In an optimal parallel build, all agents are busy at the same time.

For example, the image on the left shows a build that suffers from over serialization. Using the Serialization analysis report, you can determine why the build is serialized and adjust the build to eliminate serializations. The image on the right shows the result of making the correct adjustments.



- Watch for the following in the Agents & Jobs section:
  - One agent bar is longer than others
  - If you see gaps where an agent is not running a job, carefully examine the running jobs while some agents are idle
- Conflicts and reverted jobs

The presence of conflict and reverted jobs means the build was slower than necessary. Reverted jobs are symptoms of job conflicts. To eliminate reversions, you must eliminate conflicts. Usually, running a build with a history file eliminates conflicts. Typically, the most expensive job conflicts are those involving parse jobs.

If your build ran without a history file, you may have conflicts. If you had a pre-existing history file, your build should not contain conflicts unless the build changes affected job order and inter-dependencies.

## Appendix A: Annolib Programmer's Reference

annolib is the library used to parse annotation files. Required values are enclosed in < > (for example <filename>). Optional values are enclosed in ? (for example ?count?).

- `anno create` - Create a new anno object that can be used to query information from an annotation file. The return value is a handle which can be used in a subsequent [`$anno ...`] call. For example:  

```
set anno [anno create]  
  
$anno agents
```
- `$anno agents` - Retrieve a list of agents that participated in this build.
- `$anno comparejobs <field> <jobId> <jobId>` - Compare two jobs for sorting by the given field, returning -1, 0, or 1 if the first job is earlier, equal to, or later than the second job.
- `$anno destroy` - Release all resources associated with this anno instance and remove the command for controlling it from the interpreter.
- `$anno duration` - Retrieve the length of the build described in the anno object. This is the greatest time index seen in the annotation file.
- `$anno environment` - Dump the contents of the environment table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to [`array get`]. The list is unordered.
- `$anno file <command> <filename>` - Query the anno object for information about a file referenced by the build.
- `$anno file isdirectory <filename>` - Return a boolean indicating whether the specified file is a directory or not.
- `$anno file operations <filename>` - Return a Tcl list of lists describing the operations that referenced this file in the build.
- `$anno file type <filename>` - Return the file type for the specified file.
- `$anno files` - Retrieve a list of the files referenced in Operations in this annotation file.
- `$anno filecount` - Retrieve a count of the number of files referenced in Operations in this annotation file.
- `$anno indexagents` - For each agent in the annotation, construct a list of jobs run on that agent, sorted in order of time they were invoked. This index is stored internally, for use by ElectricInsight when rendering the annotation.

- `$anno load <channel> ?count?` - Parse `?count?` bytes of data from `<channel>`, creating job objects as necessary. The return value is a boolean indicating whether or not parsing is complete. If `?count?` is not specified, all data will be read from `<channel>`.
- `$anno loadstring <data> ?done?` - Parse the data in `<data>`, creating job objects as necessary. There is no return value. If `?done?` is specified, it indicates whether or not the string represents the end of the XML data.
- `$anno jobcommand <jobId>` - Invoke commands on a particular job from the annotation.
- `$anno jobagent <jobId>` - Retrieve the name of the agent the job was run on.
- `$anno job annolength <jobId>` - Retrieve the length of the segment of the annotation file describing this job.
- `$anno job annostart <jobId>` - Retrieve the start of the segment of the annotation file describing this job.
- `$anno job commands <jobId>` - Retrieve a list of the commands associated with this job. The result is a list of lists with the form:

```
cmdlist = {[cmd]}
cmdlist is a list containing one or more cmd
cmd = {lines argv [output]}
cmd is a list containing lines, argv, and one or more output
lines = the actual line(s) from the makefile from where the command was issued, for example "11-12"
argv = the actual command issued to the system
output = {src out}
output is a list containing src and out
src = the source of the output, e.g. "make"
out = the actual string emitted by the component
```

- `$anno job conflict <jobId>` - Return registry conflict information for the specified job.
- `$anno job conflictfile <jobId>` - If this job is a conflict job, retrieve the name of the file that it conflicted over.
- `$anno job conflicttype <jobId>` - If this job is a conflict job, retrieve the type of the conflict.
- `$anno job deps <jobId>` - Retrieve the list of history dependencies for this job. This may be empty if annotation did not include history-level detail, or if there were no history dependencies.
- `$anno job environment <jobId>` - Retrieve environment for the specified job.
- `$anno job exitcode <jobId>` - Retrieve the exit code for this job.
- `$anno job finish <jobId>` - Retrieve the end time of the job in seconds.
- `$anno job flags <jobId>` - Retrieve the flags for the job as a Tcl list.
- `$annojob isconflict <jobId>` - Return a boolean indicating whether or not this job is a conflict job.
- `$anno job isrerun <jobId>` - Return a boolean indicating whether or not this job is a rerun job.
- `$anno job isreverted <jobId>` - Return a boolean indicating whether or not this job is a reverted job.
- `$anno job length <jobId>` - Retrieve the duration of the job in seconds. This is equivalent to `[$anno job jobId finish] - [$anno job jobId start]`, but it avoids using Tcl's `[expr]` command, so it is much faster.
- `$anno job make <jobId>` - Retrieve the Make instance ID for the Make containing the specified job.

- `$anno job makefile <jobId>` - Retrieve the name of the makefile containing the rule that produced this job, if any.
- `$anno job name <jobId>` - Retrieve the name for the specified job.
- `$annojob neededby <jobId>` - Retrieve the ID of the job that caused this job to be run.
- `$anno job operations <jobId>` - Retrieve the list of Operations performed by this job, in order, as a Tcl list of lists.
- `$anno job operations -type registry <jobId>` - Retrieve the list of registry operations for the specified job.
- `$anno job partof <jobId>` - Retrieve the ID of the job that this job continues. Only valid for continuation jobs; other jobs will return an empty string.
- `$anno job rerunjob <jobId>` - For conflict jobs, retrieve the job that was used to rerun the job.
- `$annojob serialorder <jobId>` - Retrieve the serial order of the job relative to the other jobs in the build; the first job in the build has serial order 1; the final job in the build has serial order N for a build with N jobs.
- `$anno job start <jobId>` - Retrieve the start time of the job in seconds.
- `$anno job timing <jobId>` - Retrieve the full timing information for this job as a list of lists of the form {start finish agent} {start finish agent} .... Normally there will be only one entry in the list; if the job was restarted due to cluster sharing or agent failure, there will be additional entries. Jobs that never ran will return an empty list.
- `$anno job type <jobId>` - Retrieve the type of the specified job.
- `$anno job submakes <jobId>` - Retrieve the list of submakes performed by the specified job.
- `$anno job waitingjobs <jobId>` - Retrieve the list of jobs that had to wait for the job to complete before running. This may be empty if there are no waiting jobs, or if the annotation file did not include waitingJobs annotation.
- `$anno job writejob <jobId>` - For conflict jobs, retrieve the job that wrote the file that the job conflicted over.
- `$anno jobcount` - Return a count of the number of jobs in the annotation.
- `$anno jobiterbegin` - Initialize the job iterator to the head of the list of jobs in the anno object.
- `$anno jobitermore` - Return a boolean indicating whether the next call to [`$anno jobiternext`] will return a valid job or not. This is used in conjunction with [`anno jobiterbegin`] and [`anno jobiternext`] to efficiently iterate through the list of jobs in the anno object, in serial order.
- `$anno jobiternext` - Retrieve the job with the next highest serial order in the anno object using an iterator initialized with [`anno jobiterbegin`]. If the iterator has reached the end of the list of jobs, an empty string is returned; otherwise the job ID for the next job is returned, and the iterator is advanced one step.
- `$anno jobsearch <attribute pattern>` - Search the jobs in the annotation for jobs that match the given criteria.
- `$anno make <command> <makeId>` - Invoke commands on a particular Make from the annotation.
- `$anno make commandline <makeId>` - Retrieve the command line for the specified Make.
- `$anno make job <makeId>` - Retrieve the job ID for the job that spawned the given Make instance, or an empty string if no job spawned the Make.

- `$anno make level <makeId>` - Retrieve the level of the specified Make instance.
- `$anno make mode <makeId>` - Retrieve the emulation mode of the given Make instance.
- `$anno make workingdir <makeId>` - Retrieve the working directory for the specified Make instance.
- `$anno makecount` - Return a count of the number of Makes in the annotation.
- `$anno metrics` - Dump the contents of the metrics table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to `[array get]`. The list is unordered.
- `$anno parseoptions ?optionList?` - Query or set the anno object parse options. This controls which portions of the annotation file are processed when `[anno load]` is invoked.
- `$anno properties` - Dump the contents of the properties table, which was built out of the section of the annotation file. The result is a list of name/value pairs similar to `[array get]`. The list is unordered.
- `$anno refcount` - For testing only. Retrieve the reference count from the anno object.
- `$anno rjobiterbegin` - Initialize the job iterator to the end of the list of jobs in the anno object.
- `$ann orjobitermore` - Return a boolean indicating whether the next call to `[anno rjobiternext]` will return a valid job or not. This is used in conjunction with `[anno rjobiterbegin]` and `[anno rjobiternext]` to efficiently iterate through the list of jobs in the anno object, in reverse serial order.
- `$anno rjobiternext` - Retrieve the job with the next lowest serial order in the anno object using an iterator initialized with `[anno rjobiterbegin]`. If the iterator has reached the end of the list of jobs, an empty string is returned; otherwise the job ID for the next job is returned, and the iterator is advanced one step.
- `$anno sortjobs ?options? <jobList>` - Sort the jobs in `<jobList>` according to the given criteria. WARNING: This modifies the list *in place*.
- `$anno type <command> <type>` - Query the aggregate attributes of jobs by type.
- `$anno type conflicttime <type>` - Return the amount of time spent on jobs of this type that were later determined to be in conflict.
- `$anno type jobcount <type>` - Return the number of jobs of this type.
- `$anno type reruntime <type>` - Return the amount of time spent on jobs of this type that were rerun jobs.
- `$anno type revertedtime <type>` - Return the amount of time spent on jobs of this type that were later reverted.
- `$anno type time <type>` - Return the amount of time spent on jobs of this type that were not conflict, reverted, or rerun jobs.
- `$anno types` - Retrieve a list of known types of jobs.